Universidad Autónoma de Madrid
Escuela politécnica superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

DESARROLLO DE UN LENGUAJE DE EXPERIMENTACIÓN PARA UNA PLATAFORMA DE ADQUISICIÓN DE ODORANTES

Autor: Ángel Fuente Ortega
Tutor: Francisco de Borja Rodríguez Ortiz

Mayo 2017

DESARROLLO DE UN LENGUAJE DE EXPERIMENTACIÓN PARA UNA PLATAFORMA DE ADQUISICIÓN DE ODORANTES

Autor: Ángel Fuente Ortega
Tutor: Francisco de Borja Rodríguez Ortiz

Grupo de Neurocomputación Biologica (GNB)
Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Mayo 2017

Resumen

Resumen

Actualmente la clasificación de gases puede ser de gran utilidad en diferentes ámbitos, como medir la calidad del aire o detectar el número de personas en una habitación. El tamaño de los sensores de odorantes, conocidos como narices electrónicas, y detectores de gas se ha reduciendo progresivamente dando lugar a sensores más pequeños, pudiendo crear varias plataformas de tamaño más reducido y versátil para la experimentación, usando para ello sensores de bajo coste y sistemas embebidos, como un BeagleBone en este proyecto.

La plataforma que se ha utilizado es de tamaño reducido, y está realizada con elementos low-cost. Esta plataforma se encuentra en una fase de prototipado y tiene un sistema de control básico, por lo que el objetivo principal de este proyecto consiste en la creación de un software de control, junto con un pseudo-lenguaje para que la interacción con la plataforma sea simple y poder explorar al máximo el funcionamiento de esta.

El sistema está diseñado e implementado para controlar la plataforma y realizar experimentos de forma sencilla. Con este software se puede ejecutar los distintos modos de captación de odorantes sin necesidad de reescribir el código. De esta forma puede ser utilizado por cualquier persona que quiera experimentar sin necesidad de que conozca su funcionamiento interno.

Con el desarrollo del protocolo se ha conseguido realizar varios experimentos de forma simple, sin necesidad de profundizar en los elementos hardware y sin tener que modificar continuamente el software. Para demostrar el correcto funcionamiento de la plataforma y del software de control, se han realizado varios experimentos y se ha analizado los resultados.

Palabras Clave

Nariz artificial, software de control, captación de odorantes, sensor MOS, Python, Beagle Bone, Figaro 2600, modulación en amplitud, modulación temperatura, modulación frecuencia

Abstract

Currently, odorant classification can be very useful in different areas, like measuring air quality or detecting the number of people in a room. These sensors known as electronic noses and gas detectors have been progressively reduced resulting in smaller platforms versatile for experimentation. In this proyect we have used low cost sensors and embedded systems such as BeagleBone.

The platform that has been used is of reduced size, made with low cost elements. The platform is in a prototyping phase and has a basic control system, so the main goal of this proyect is to create a control software, along with a pseudo-language so that the interaction with the platform is simple. This would be oriented to the automatic classification of odorants

The system has been designed and implemented to control the platform and easily perform the experiments. With this software many differents modes of odorant captures without rewriting the code. Therefore it can be used by anyone with having to know its internal operation.

The developed protocol allows to perform several experiments in a simple way, without deepening in the hardware elements that are part of the platform or having to continuously modify the software. In order to demonstrate the correct functionong of the platform and the control software, several experiments have been carried out and the results analyzed.

Key words

Artificial nose, control software, odorant capture, MOS sensor, Python, Beagle Bone, amplitude modulation, temperature modulation, frequency modulation

Agradecimientos

Quiero dar las gracias a mi familia primero, por haberme apoyado y haber confiado en mí, pagando los costes permitiendo que pudiese estudiar la carrera. También a mi tutor Francisco, por haberme guiado en este TFG y por su ayuda para acabarlo.

También agradecer a mis amigos y compañeros por haberme ayudado cuando he tenido problemas y me han ayudado y han estado allí a las buenas y a las malas, sobre todo a Mario, Carlos, Irene, Manu y Roy. También agradecer a mis amigos de la carrera Oscar,Silvia,Patricia y Alejandro y a mís amigos del instituto y del colegio, sobre todo a Borja y David.

Índice general

Índice de Figuras

Índice de Tablas

Glosario

- **BBB**: Beagle Bone Black
- **GNB**: Grupo de Neurocomputación Biológica
- **Drift**: Obtener distintos valores en las mismas condiciones debido a la memoria del sensor
- **ADC**: Analog to Digital Converter
- **PWM**: Pulse With Modulation
- **GPIO**: General Purpose Input/Output
- **Captura**: Medición de los 4 odorantes en una secuencia determinada
- **SO**: Sistema operativo

1 Introducción

1.1. Motivación del proyecto

El siguiente Trabajo de Fin de Grado se centra en la creación de un pseudo-lenguaje para la ejecución de experimentos en una plataforma de captación de odorantes que tiene el Grupo de Neurocomputación Biológica (GNB), así como su validación con la realización de varios experimentos con distintos algoritmos de captura de odorantes.

El sistema olfativo es capaz de captar una gran cantidad de odorantes. Estos son captados en los receptores que se encuentran en la cavidad nasal. El olor, según la RAE, es la impresión que los efluvios producen en el olfato (Española, 2010). Un olor es una combinación de distintos gases, que según la composición influyen más o menos en la percepción del olfato.

Actualmente hay una gran cantidad de sensores en distintos tipos de entornos, como por ejemplo en sensores de humo, equipos médicos o detección de escapes en estaciones, pero pueden utilizarse en otros entornos como detección de estupefacientes en aeropuertos o monitorización del número de personas en una sala a partir del nivel de CO_2.

Para tratar de imitar el sistema olfativo humano, se han desarrollado sensores con el mismo fin, tratando de detectar y discriminar distintos odorantes. Estos sensores se conocen como narices artificiales o Electronic Nose (EN). En la actualidad, el GNB tiene desarrollado el hardware de una EN, pero esta se encuentra en una fase de inicial, es un prototipo y el software de control que tiene es muy básico.

Este proyecto parte de la base de otros trabajos anteriores del GNB (Gutiérrez et al., 2016; Villarreal, 2009; Yáñez et al., 2012). El propósito de este trabajo es el de crear un software versátil, que permita realizar experimentos de forma fácil, permitiendo que cualquier persona ajena a la plataforma sea capaz de utilizarla.

Para este trabajo se utilizará la plataforma de captación que tiene el GNB (Gutiérrez et al., 2016). Se adaptará el software de control básico de la plataforma y se implementará un software de control nuevo para que la plataforma sea fácil de utilizar. Además de la realización del pseudo-lenguaje para el protocolo de experimentación, se realizarán varias pruebas para comprobar que el código realizado funciona correctamente y comprobar que el pseudo-lenguaje es correcto, y permite el control de todos los elementos de la plataforma, pudiendo realizar una gran cantidad de experimentos con las diferentes técnicas de captación de odorantes. El software que se ha

desarrollado tiene el nombre de **PyHuele**.

1.2. Objetivos

El objetivo general de este TFG es el desarrollo de un software versátil y fácil de utilizar para la plataforma de experimentación, desarrollando un pseudo-lenguaje para la adquisición de odorantes sin necesidad de saber programar o entender el funcionamiento de la plataforma. Para su desarrollo se estudiarán los parámetros que se usan en la configuración de los experimentos.

Los sub-objetivos derivados de este objetivo principal, que son necesarios para poder realizar correctamente el proyecto, son los siguientes:

- Estudio, análisis y entendimiento de la plataforma de captación de odorantes (Gutiérrez et al., 2016):
  - Los circuitos de los que están compuesto la plataforma, y que componen los distintos tipos de algoritmos de captación de odorantes.
  - Entender los códigos primitivos de captación de los distintos odorantes.
- Lectura y compresión de publicaciones sobre narices electrónicas y distintos tipos de modulación.
- Creación de un software de control y pseudo-lenguaje para el control de la plataforma y la realización de experimentos.
- Realización de scripts para visualizar los datos.
- Validación del software de control y estudio de los resultados obtenidos mediante la realización de experimentos en la plataforma.

1.3. Estructura de la memoria

La memoria de este TFG se encuentra dividida en distintas secciones:

- **Estado del arte:** En esta sección se describe el marco teórico que es necesario entender para realizar el proyecto. Los tipos de sensores olfativos, la plataforma de captación de odorantes, el proceso de transformación de un odorante en un pulso eléctrico y la explicación de los distintos tipos de modulación.
- **Sistema y diseño:** Se explica el pseudo-lenguaje de experimentación desarrollado, tanto las palabras claves como la notación para realizar los experimentos y los módulos que componen el software de control. También se muestran los requisitos funcionales y no funcionales y los diagramas funcionales y de casos de uso.
- **Desarrollo:** Se explica todo el desarrollo software de control llevado a cabo a lo largo del proyecto.
- **Validación del software de control y pseudo-lenguaje:** Se realizarán varias pruebas para comprobar que el pseudo-lenguaje está completo y que se puede controlar todos los elementos de la plataforma. También se muestran los resultados de los experimentos con los diferentes tipos de algoritmos de captura comentando los resultados.

2

Estado del arte

En este capítulo se enuncian las bases teóricas para entender el trabajo realizado: los diferentes tipos de sensores olfativos, el funcionamiento de los sensores y de la plataforma de captación, y los algoritmos usados para la captación de odorantes.

2.1. Introducción

En la actualidad hay varios problemas relacionados con la detección de olores: detección de partículas nocivas en el ambiente, uso en equipos médicos, contabilizar el número de personas en una sala donde no puede ponerse una cámara, discriminación de odorantes, etc.

2.2. Sensores olfativos

En la actualidad existen varios tipos de sensores olfativos que pueden utilizarse. A continuación se exponen algunos de los principios físicos en los que se basan los sensores olfativos más comunes (Pearce et al., 2006):

- **Quimiorresistores**: sensores que varían la conductividad eléctrica dependiendo del elemento químico al que se le exponga. Son sensores poco selectivos siendo más complicado la detección de un odorante, pero tienen un tamaño y un precio más reducido. Los sensores MOS que se utilizan en la plataforma son de este tipo.
- **Quimiotransistores**: transistores con un recubrimiento que los hace sensibles a sustancias químicas. Pueden usarse en entornos líquidos y gaseosos.
- **Quimiocapacitores**: sensores formados por un condensador que tienen un dieléctrico formado por un polímero que varía de capacidad cuando absorbe o libera moléculas gaseosas.
- **Quimiosensores amperimétricos**: miden la corriente que pasa por una celda electroquímica, siendo muy sensible y muy selectivos, pero con un coste elevado.

Principio	Medida	Tipo sensor
Conductividad eléctrica	Conductancia	MOS y químico resistivo
Capacidad	Capacitancia	quimio capacitivos por polímeros
Potenciométrico	Voltaje/emf y I-V/C-V	Schottky Diode o MOSFEST
Amperimetría	Corriente	Electro catalítico
Calorímetro	Temperatura	Químico Térmico - Termistor, Termopar
Óptico	Intensidad/Espectro	Fluorescencia y quimioluminiscencia

Cuadro 2.1: Clasificación de distintos sensores olfativos

En la tabla 2.1, se muestran algunos de los tipos de sensores más utilizados, el principio físico en que se basan, la cualidad física que miden y el tipo de sensor (Pearce et al., 2006).

El tipo de sensor utilizado en la plataforma es un sensor quimiorresistivo de tipo MOS (Metal Oxide Semiconductor), que está compuesto de una base sobre la que se encuentra el material de detección.

2.3. Propiedades eléctricas de un gas en un circuito cerrado

Para que se pueda captar la señal que produce un gas, es necesario que se consiga obtener una señal eléctrica de este. El funcionamiento de un sensor quimiorresistivo es el siguiente:

- El sensor, en presencia del aire no contaminante, genera una resistividad muy alta, impidiendo el paso de la corriente por sí mismo. Esto se debe a que el oxígeno se acumula en la superficie del sensor, provocando que los electrones se vean atraídos por las partículas de oxígeno, impidiendo el libre flujo de electrones. Este proceso se observa en el panel de la izquierda en las figuras 2.1 y 2.2.

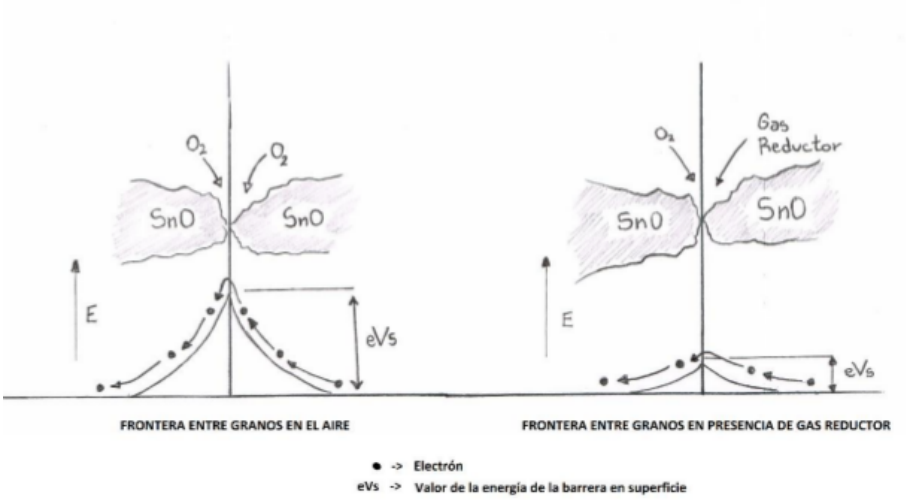


Figura 2.1: Interacción del sensor con el ambiente. A la izquierda en aire libre, a la derecha en presencia de un gas contaminante. Imagen adaptada de (Figaro, 2016a) obtenida de (Gutiérrez et al., 2016)

- Cuando se expone el sensor a un gas, se oxida la superficie del sensor y las partículas del gas eliminan a las de oxigeno. Al eliminarse las partículas, el potencial de barrera que crearon

las partículas de oxígeno disminuye, provocando un aumento del flujo en los electrones y disminuyendo la resistividad del sensor. Este proceso se observa en el panel de la derecha en las figuras 2.1 y 2.2.

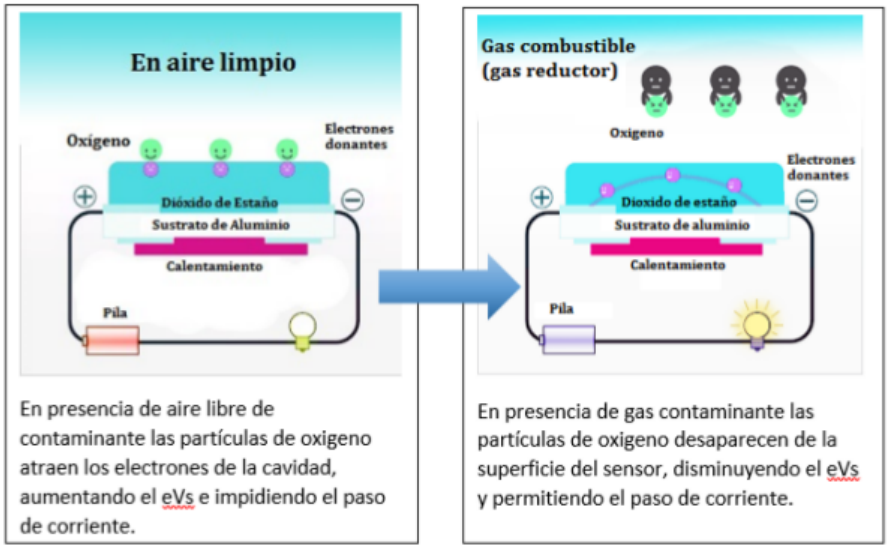


Figura 2.2: Comportamiento del sensor. Imagen adaptada de (TGS, 1978) obtenida de (Gutiérrez et al., 2016)

Como consecuencia del funcionamiento del sensor, se puede usar la resistencia como medida para la obtención directa de información sobre el gas que se está analizando y su concentración.

2.4. Plataforma de experimentación

La plataforma que se va a utilizar como base para el desarrollo del pseudo-lenguaje y el software de control se compone de un total de 5 circuitos distintos. El controlador de la plataforma es una BeagleBone Black(BBB), que es un sistema embebido lo suficiente potente para proporcionar la corriente necesaria a los circuitos que se usan y con los suficientes pines para obtener la información de los distintos circuitos. Los diferentes circuitos que componen la plataforma son:

- Circuito de adquisición de odorantes con temperatura constante.
- Circuito de adquisición de odorantes con temperatura variable y codificación en amplitud.
- Circuito de adquisición de odorantes con temperatura variable y codificación en frecuencia.
- Circuito para el control de las electroválvulas.
- Circuito para el control del motor de succión.

La plataforma funciona de la siguiente forma: el odorante se encuentra en unos botes, que se sitúan en las zonas habilitadas para ello. Cuando se activa el motor, este succiona el odorante, que viaja por los tubos, pasando a través de las electroválvulas que se encuentren abiertas y llegando al sensor. El sensor genera una señal eléctrica en su interior (sección 2.3) y esa señal llega al circuito ADC, un conversor analógico digital de la BBB, que la registra. En la figura 2.3 se muestra el aspecto de la plataforma. Otro ejemplo de plataforma de captación de odorantes

es la que se encuentra en Extremadura (Macías et al., 2013, 2012), aunque a diferencia de la plataforma del GNB esta no consta de diferentes modos de variación de la temperatura del sensor.

Los diferentes elementos que pueden programarse en la plataforma son y que son controlados por el proyecto realizado son:

- La potencia del motor para aumentar o disminuir su capacidad de succión. Este se controla usando los puertos PWM (Pulse With Modulation).
- Abrir o cerrar las electroválvulas. Usando los puertos GPIO (control digital).
- Regular la temperatura a la que se calienta el sensor. Para su control se utiliza un puerto PWM.
- Programar diversos parámetros de las ejecuciones para variar el tiempo de captura de un odorante o la temperatura de calentamiento del sensor entre otros.

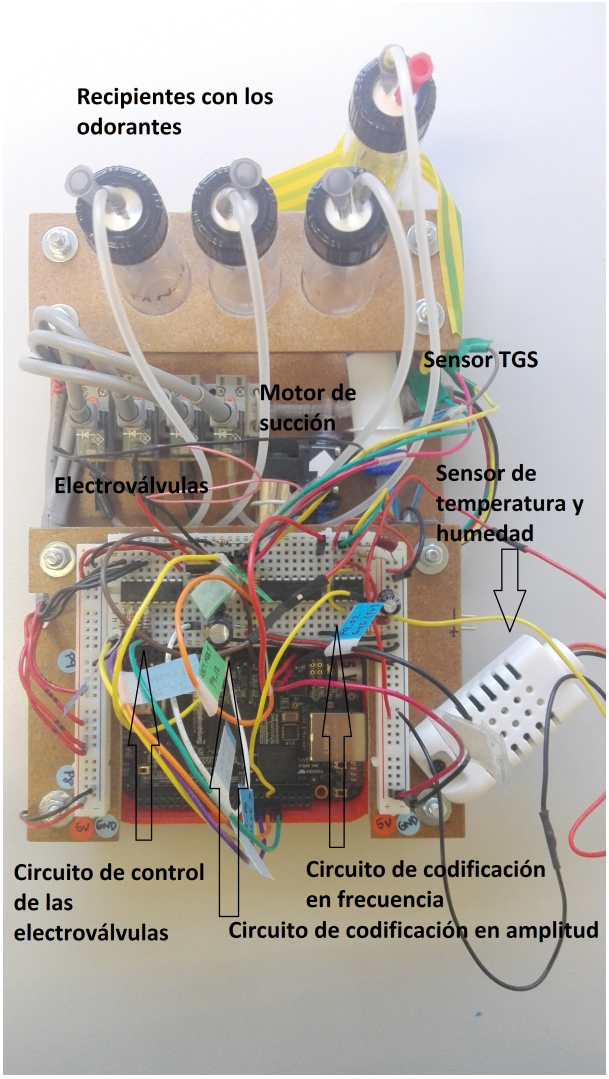
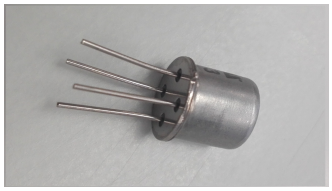


Figura 2.3: Plataforma de captación de odorantes

2.5. Sensor TGS2600

El sensor utilizado en la plataforma es el TGS2600, un sensor quimioresistivo. El sensor tiene cuatro patillas de salida y una carcasa exterior que lo protege de la suciedad del ambiente como se muestra en la figura 2.4 en el panel de la izquierda. En el panel de la derecha se muestra un esquema del sensor con los elementos que lo componen.



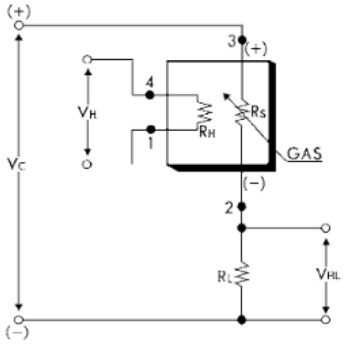


Figura 2.4: Figura del sensor TGS2600 a la izquierda y esquema del sensor TGS2600 a la derecha

El sensor está compuesto por dos resistencias, una está conectada a los pines 1 y 4 y es la encargada del calentamiento, regulando la temperatura (resistencia R_H). El voltaje que se aplica en estos pines es el voltaje de calentamiento V_H. Para este tipo de sensor el voltaje que indica el fabricante es 5V (Figaro, 2016b). Sin embargo, esta plataforma consta de varias técnicas que varían este voltaje, obteniendo diferentes modos de operación (Lee and Reedy, 1999). La resistencia conectada a los pines 2 y 3 varia según los cambios del potencial de barrera (sección 2.3).

La resistencia R_L es una resistencia de carga que se conecta en serie con el sensor, que se utiliza como un divisor de tensión para obtener el valor medido por el sensor V_{HL} (Figaro, 2016b).

2.6. Técnicas de medición de la plataforma

Actualmente en la plataforma se tienen tres modos o técnicas de operación con los que se captan los odorantes, aunque en la actualidad hay más métodos (Gutierrez-Osuna and Hierlemann, 2010). Estos modos se dividen según la temperatura de calentamiento del sensor a la que se capta el odorante, dividiendo las técnicas en dos grupos: los de temperatura constante y temperatura variable. Con la variación de la temperatura se busca cambiar las propiedades físicas del sensor, cambiando así la forma de medir del sensor (Herrero-Carrón et al., 2015; Vergara et al., 2007).

2.6.1. Adquisición de odorante con temperatura constante

En este tipo de técnica no se aplica ningún cambio ni se filtra la señal, simplemente se recoge la información tal y como se capta del sensor. A este tipo de modulación se le llamará puro.

Actualmente esta temperatura se encuentra siempre al máximo, pero se puede cambiar de forma que varíe de forma constante, siguiendo un patrón constante y ver el efecto que produce, como se propone en (Hossein-Babaei and Amini, 2014), que usando la inducción por choque térmico consigue una distinción casi perfecta de diferentes odorantes usados.

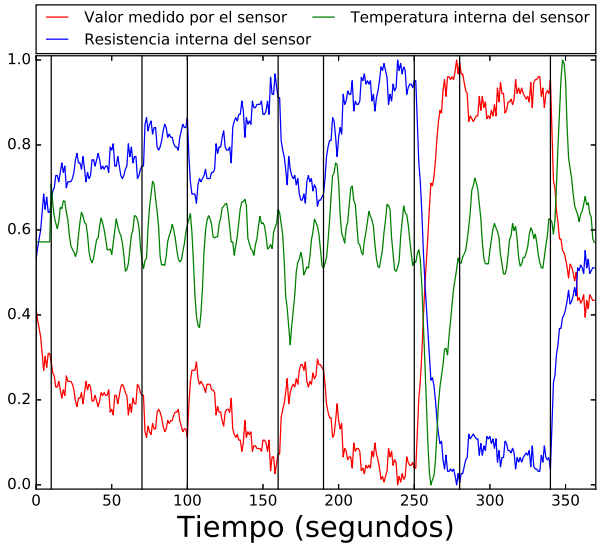


Figura 2.5: Gráfica de una captura siguiendo el temperatura variable y codificación en amplitud. La secuencia de gases es: muestras iniciales, aire, metanol, aire, etanol, aire, butanol y aire. Cada linea vertical representa un cambio de gas. Esta captura se ha realizado con la concentración #4 (tabla F.1). Script de ejecución en anexo G.1.3

2.6.2. Adquisición de odorantes mediante temperatura variable: codificación en amplitud

Este modo de adquisición de odorantes, al contrario que las técnicas tradicionales que fijan un perfil de calentamiento de la temperatura, como por ejemplo una señal sinusoidal y leen el valor sensor (Gutierrez-Osuna et al., 2003), ajusta la temperatura basándose en los valores previos. Con esta interacción entre el gas y el sensor se consigue que la señal varíe dinámicamente, obteniendo así los patrones de conductividad mínimos y máximos, que son característicos de cada gas (Herrero-Carrón et al., 2015; Vergara et al., 2005).

Con la auto-regulación propia del sistema se consigue que la señal no sature, ya que cuando va a llegar al límite, máximo o mínimo, el sistema ajusta la señal de la temperatura, aumentándola en el caso de que se vaya a sobrepasar el límite inferior o disminuyéndola en el caso contraria, evitando así la saturación de la señal. En la figura 2.5 el valor verde se corresponde con la temperatura del sensor.

De esta forma, la temperatura se regula con cada medición que se captura. La fórmula que se utiliza para calcular el nuevo valor de la temperatura del sensor es:

$$T_{heat\_new} = T_{heat\_old} - (SLOPE * TEDENCIA)$$

.

En la fórmula, la variable SLOPE es el resultado de aplicar una regresión lineal a los valores de las medidas previas, y la variable TENDENCIA determina la amplitud en el cambio de la temperatura.

Como se explicó en el apartado 2.3, los valores de la resistividad del sensor y el valor eléctrico obtenido de este son opuestos, y la temperatura se modela según la fórmula expuesta anteriormente. Por lo que cuando el valor de salida del sensor baja, la temperatura sube, y viceversa. Este tipo de modulación es la que se usa en la plataforma, y se la llamará regresión.

2.6.3. Adquisición de odorantes mediante temperatura variable: codificación en frecuencia

El método de captación de odorantes que se explica en este apartado también varía la señal de la temperatura, pero en frecuencia en vez de en amplitud. Este método es desarrollado por el profesor Martinelli (Martinelli et al., 2012, 2013). En estos trabajos, se expone que el efecto que un odorante produce en el sensor depende de la temperatura de este, y que cada odorante tiene una sensibilidad a la temperatura propia.

Para lograr este método de captura de odorante, Martinelli propone un circuito cerrado, donde la salida del sistema se utiliza como señal de entrada para el calentamiento del sensor, figura 2.6. El circuito cerrado se compone de dos bloques, el primero consta del sensor y una interfaz electrónica, para captar y procesar la señal y el segundo bloque se compone de una interfaz que procesa la señal de salida (Xout), para obtener la señal de calentamiento adecuada.

El circuito que se utiliza es un multivibrador astable, donde la resistencia del sensor influye en la frecuencia de la señal de salida. La elección de este circuito se debe a que es la forma más simple de generar una señal periódica. Para implementarlo se utiliza el temporizador NE555 (Instruments, 2016), que se encarga de generar una frecuencia de reloj electrónica en forma de onda cuadrada, a la que denominaremos pulsos, figura 2.7. Esta señal se registra y se utiliza en la interfaz de temperatura.

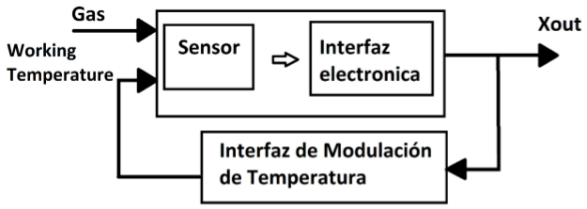


Figura 2.6: Diagrama de bloques propuesto para la adquisición de odorantes mediante temperatura variable y codificación en frecuencia. Figura adaptada de (Martinelli et al., 2012)

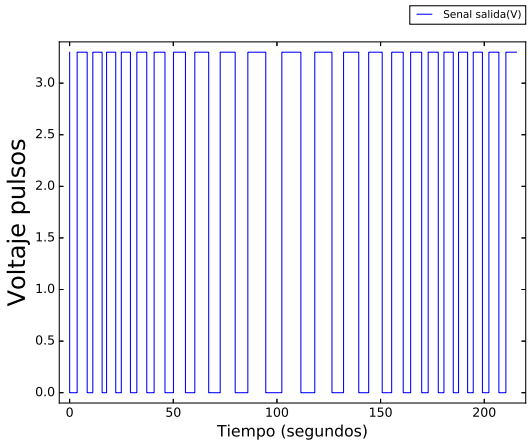


Figura 2.7: Señal de salida. Imagen obtenida de las capturas de la plataforma

La resistencia del sensor varia el ritmo de los pulsos generados por el temporizador NE555. Cuanto mayor sea la resistencia, menos oscila el circuito y más tarda en realizar los pulsos y viceversa. Según el odorante introducido en el circuito, se producirá un cambio mayor o menor en la resistencia del sensor y por lo tanto en la velocidad de oscilación del circuito.

Estas variaciones en la resistencia del sensor, y por lo tanto en la velocidad a la que se realizan los pulsos, condicionan la temperatura del sensor.

La interfaz de modulación térmica se compone de un contador software, que tras registrar un número de pulsos produce una transición entre los dos estados lógicos del contador (estado bajo y estado alto). Para representar la transición entre los dos estados lógicos se utilizan 16 pulsos, como expone Martinelli en su trabajo (Martinelli et al., 2012). Cada 8 pulsos se pasa de un estado al siguiente, variando la temperatura del sensor en cada estado. En la figura 2.8 se puede ver como en el primer estado la temperatura disminuye sincronizada con los pulsos y en el segundo estado aumenta. Estos valores que se usan no son los óptimos, pudiendo haber un número de pulsos mejor.

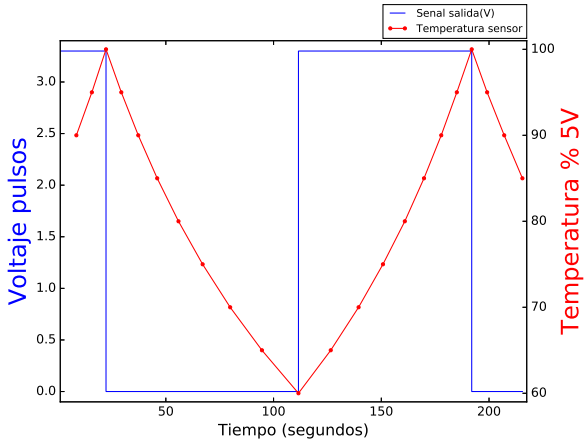


Figura 2.8: Transición entre dos estados lógicos y la temperatura del sensor. Imagen obtenida de las capturas de la plataforma

En la figura 2.9 se muestra la transición de los estados lógicos junto a los pulsos que produce el circuito oscilador y la temperatura.

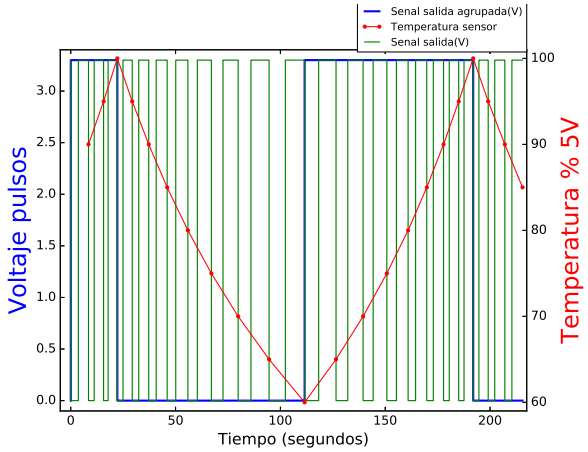


Figura 2.9: Representación de los pulsos producidos por el circuito, junto con la transición entre ambos estados y la temperatura del sensor. Imagen obtenida de las capturas de la plataforma

3

Sistema y diseño

En este capítulo se detalla el software de control desarrollado para la plataforma. El pseudo-lenguaje desarrollado es muy simple y está compuesto por las características del experimento seguido del valor de esas características. Por ejemplo, una característica configurable dentro del lenguaje es la potencia de succión del motor y un valor asociado seria 70, que hace referencia al 70 % del valor máximo, como se muestra en G.

3.1. Módulos del pseudo-lenguaje de experimentación y del software de control

El software de control está divido funcionalmente en tres módulos diferentes. Los módulos que se han desarrollado son:

- Módulo de recogida y tratamiento de datos.
- Módulo de captura de datos.
- Módulo de representación y visualización.

3.1.1. Módulo de recogida y tratamiento de datos

Este módulo se encarga de recoger los datos relacionados con las capturas y de procesarlos para que todo esté correcto. La forma de introducir los datos es a través de un fichero de texto plano, un script, donde se escriben todas las características que se desea que tenga el experimento y los valores de esas características. El orden en que se escriben los datos es irrelevante.

Una vez que se le pasa a **PyHuele** el script con la información acerca de las capturas, esta se trata. Para realizar varios experimentos, se replica la información y se crean varias listas con los valores de todas las ejecuciones, de forma que cada lista tiene todos los valores de una ejecución.

Si al tratar la información se encuentra un error en el script que se le ha pasado a **PyHuele**, se devuelve un error indicando el motivo y se finaliza la ejecución.

3.1.2. Módulo de captura de datos

El módulo de captura de datos está compuesto por los diferentes modos de captura de datos implementados. Las técnicas de captura de odorantes implementadas son:

- Algoritmo de captación con temperatura fija.
- Algoritmo de captación con temperatura variable y codificación en amplitud.
- Algoritmo de captación con temperatura variable y codificación en frecuencia.

Los dos primeros métodos son síncronos, cada segundo se toma una muestra de odorante, mientras que el último es asíncrono y depende de la resistencia interna del sensor.

3.1.3. Módulo de visualización de datos

Con este módulo se representan todo los datos captados. Según el modo de captura de datos escogido y la opción indicada en el script de ejecución, se representarán los datos de una forma u otra. En este módulo se agrupan todos los datos que tengan las mismas características y se representan juntos.

3.2. Pseudo-lenguaje para el protocolo de experimentación

El pseudo-lenguaje diseñado para este proyecto tiene que ser simple, ya que debería ser usado por cualquier persona, así que todos los elementos que lo conforman junto con su sintaxis, tienen que ser fáciles de entender.

3.2.1. Sintaxis del protocolo

Para la sintaxis del protocolo se ha seguido un esquema simple: una palabra clave, seguida del símbolo ':' y por último el valor de la palabra clave que se usará en la ejecución.

- {Palabra clave}:{Expresión con los valores de las ejecuciones}

La palabra clave, que hace referencia a una característica del experimento, es un elemento configurable que interviene en la captura de los datos. El símbolo ':' es un separador entre la palabra clave y la expresión con los valores que configuran la característica de la ejecución.

3.2.2. Parámetros configurables

Se ha hecho un análisis de las características que pueden configurarse y que influyen a la hora de realizar experimentos. Estas características son las que hay que configurar para ejecutar los distintos modos de captación explicados en el apartado 2.6. En un futuro, estos elementos podrían ampliarse si se quiere añadir un nuevo modo de captura o si se quiere ampliar alguno ya implementado. Estas características configurables pueden son:

- Tiempo que cada gas es analizado.
- Succión del motor.

- Gases que se van a analizar.
- Nombres de las carpetas y archivos donde se quieran guardar los datos.
- Tiempo de estabilización entre captura de gases.
- Tendencia.
- Número de muestras iniciales.
- Forma de variar la temperatura.
- Forma captar la señal en función del tiempo.
- Localización para guardar los datos.

3.2.3. Palabras clave del pseudo-lenguaje

Con las características expuestas en el apartado anterior, se han definido las siguientes palabras claves, que representan las características del experimento. Se han agrupado según la función que tienen dentro de los distintos módulos.

Modulo de tratamiento de datos

- **numero_experimentos**: indicamos a la plataforma cuantos experimentos queremos realizar. Es sólo un número.

Módulo de captura de datos

- **modulacion**: indica el modo de captura que se va a utilizar para captar los datos. Solo puede escogerse uno.
  - 1: para utilizar el algoritmo de temperatura fija.
  - 2: para utilizar el algoritmo de temperatura variable y codificación en amplitud.
  - 3: para utilizar el algoritmo de temperatura variable y codificación en frecuencia.
- **succion**: indicamos la succión del motor, en tanto por ciento. La potencia máxima del motor, que equivaldría a 100(%), es de 5V. Es muy importante que el valor introducido sea mayor que 30 y menor que 100, ambos incluidos.
- **duracion_estimulo**: representa el tiempo, en segundos o en transiciones entre estados, que se va a estar el sensor captando un gas.
- **muestras_iniciales**: el número de muestras iniciales, en segundos o en transiciones entre estados, que se quieren captar.
- **segundos_entre_estímulos**: el tiempo, en segundos o en transiciones entre estados, que se quiere dejar de aire entre gas y gas.
- **nombre_carpeta**: nombre de la carpeta donde van a guardarse las capturas. La ruta de carpetas es la siguiente: en el directorio donde se ejecuta **PyHuele**, si no está creada, se crea la siguiente estructura de carpetas: CAPTURAS/*nombre del algoritmo usado*/*Fecha de la captura*/*nombre de la carpeta*/*XSAMPLESINICIO*/, y dentro de esta última se crean 3 carpetas diferentes:

  - dat/ Donde se guardan todos los datos en un formato plano, separados por tabuladores
  - txt/ Donde se guardan todos los datos, pero con indicaciones de que es cada elemento y con una cabecera
  - TyH/ Se guardan las medidas de temperatura y humedad captadas por el sensor

- **nombre_archivo**: el nombre que se le va a dar al archivo donde van a guardarse todos los datos. Al nombre indicado se le añade la fecha y la hora del instante en que se empezó la captura de los datos.

- **tendencia**: este valor solo es necesario ponerlo en codificación en amplitud. Para el resto de los algoritmos no es necesario añadir este parámetro y si se añade, el software de control lo ignorará. Este valor representa la pendiente de calentamiento, cuanto aumenta o disminuye la señal.

- **calentamiento_sensor**: este parámetro también es específico para la codificación en amplitud. Lo que indica es la temperatura inicial de calentamiento del sensor, en tanto por ciento. El valor máximo de calentamiento es de 5V.

- **version_experimento**: indicamos a la plataforma que odorantes se quieren analizar. Hay tres versiones que se pueden poner:

  - 1 - En esta versión se pasan todos los gases a la plataforma con el siguiente orden: Olor1-Olor1-Olor2-Olor2-Olor3-Olor3-Olor4-Olor4-Olor2-Olor1-Olor3-Olor1-Olor4 -Olor3-Olor2-Olor4-Olor1. De esta forma se consiguen tener todas las transiciones entre los distintos gases.
  - 2 - Si se selecciona esta opción, los odorantes que analice el sensor se escogerán de forma aleatoria.
  - 3 - Con esta opción el usuario introduce los odorantes que desee analizar.

- **reposo**: representa el tiempo que el sistema estará en reposo, en segundos, tras realizar una captura. Esta medida se introdujo porque el motor se estropeó, y hubo que sustituirlo. No es obligatorio utilizar esta opción, es simplemente una medida de prevención.

- **vector_apertura_valvulas**: si se ha escogido en **version_experimento** la versión 3, escogemos los gases que queremos analizar. Para indicar los odorantes que se desean analizar:

  - Para el odorante 1, ponemos un 1.
  - Para el odorante 2, ponemos un 2.
  - Para el odorante 3, ponemos un 3.
  - Para el odorante 4, ponemos un 4.

- **numero_muestras**: este indicador hace referencia al número de muestras aleatorias que quieren analizarse, solo es necesario usarse cuando se indica el valor 2 en **vector_apertura _valvulas**.

- **modo_martinelli_ejecucion**: si usamos el algoritmo de temperatura variable y codificación en frecuencia, hay dos modos de ejecución, ambos relacionados con la temperatura a la que se captan las muestras:

  - 1 - Con esta opción se selecciona el modo de captura usando la ventana de tiempo.

  - 2 - Con esta opción se selecciona el modo de captura usando como referencia las transiciones entre estados. Si se usa este modo de captura, en la opción de **duracion_estimulo**, se hará referencia a las transiciones entre estados.

- **modo_martinelli_temperatura**: si usamos el algoritmo de temperatura variable y codificación en frecuencia, con este modo se seleccionará de que forma se varia la temperatura del sensor:

  - 1 - Con esta opción se selecciona el modo de cambio de temperatura de forma abrupta, cambiando de la temperatura mínima a la máxima de golpe.
  - 2 - Con esta opción se selecciona el modo de cambio de temperatura gradual, en este caso la temperatura irá aumentando o disminuyendo de forma gradual hasta llegar al máximo o al mínimo.

- **temperatura_minima_martinelli**: este valor representa la temperatura mínima que va a utilizarse con el algoritmo de temperatura variable y codificación en frecuencia.

- **temperatura_maxima_martinelli**: este valor representa la temperatura máxima que va a utilizarse con el algoritmo de temperatura variable y codificación en frecuencia.

Módulo de representación de datos

- **representar**: para indicar las opciones de representación de los datos.

  - 1- Los distintos elementos de una captura(temperatura, resistencia interna, etc) en una gráfica.
  - 2- Imprimir el mismo elemento de distintas capturas en la misma gráfica, para realizar comparaciones.
  - 3- Los elementos de una captura(temperatura, resistencia interna, etc) representados en sub-gráficas por las transiciones de los gases.

La mayoría de las palabras claves tienen relación con el módulo de ejecución, ya que son las que configuran los parámetros de los experimentos, mientras que para el módulo de representación solo se necesita indicar que modelos de representación se desea y para el de tratamiento de datos cuantas veces hay que replicar los datos.

Con estas palabras claves, se puede configurar todos los valores de la plataforma. Dependiendo del algoritmo que se utilice algunas palabras clave no se utilizan.

3.2.4. Sintaxis del pseudo-lenguaje de experimentación

Una vez explicadas las palabras claves que se han utilizado en el pseudo-lenguaje, se procede a explicar la sintaxis del protocolo.

Para la notación de las representaciones, hay que indicar los valores separados por '|', como se muestra a continuación:

- representar:1|2|3|

Una vez fijado el número de experimentos, se indican los valores de las características de los experimentos. Para separar los valores de las diferentes capturas, solamente hay que utilizar el carácter ','. Un ejemplo para 5 capturas con distintos valores de succión:

- numero_experimentos: 5
- succion: 50,60,70,80,90

En este caso hay 5 valores diferentes de succión, uno para cada captura. La primera de todas tendrá un valor de succión de 50 %, la segunda del 60 %, y así sucesivamente hasta llegar a 90 %.

Para indicar que se van a analizar varios gases en una misma captura, usando la opción de **vector_apertura_valvulas**, se usa el carácter especial '-'. Este carácter especial es un delimitador, e indica que gases por separado van a captarse en una misma captura. Un ejemplo para dos capturas distintas:

- numero_experimentos: 2
- version_experimento: 3
- vector_apertura_valvulas: 2-3-1,3-4-1-3

En este ejemplo, para la primera captura, se le indica al sistema que el orden de los gases sea: odorante2, odorante3 y odorante1. Para la segunda captura, el orden es: odorante3, odorante4, odorante1 y odorante3 de nuevo.

También se ha implementado la apertura de varias válvulas a la vez, consiguiendo de esta forma combinar los gases entre sí. Para poder combinar varios gases entre sí, hay que poner los identificadores de los gases separados por un '.'. Si, por ejemplo, se quisiese mezclar los odorantes 2 y 3 y luego analizar el odorante 1, la expresión sería la siguiente:

- vector_apertura_valvulas: 2.3-1

En resumen, con el delimitador '-', indicamos a la plataforma que gases por separado captamos, y con el '.', indicamos que queremos analizar dos o más gases a la vez.

Configuración de las repeticiones de las medidas

Como puede haber casos en los que sea necesario repetir un valor, porque quiere usarse en varias ejecuciones seguidas, se ha añadido una expresión especial, para no repetir un valor constantemente. Para repetir un valor varias veces, simplemente hay que poner detrás de ese valor lo siguiente: (*número de repeticiones*). Si se tuviesen, por ejemplo, 15 experimentos, y de todos ellos 10 tuviesen la misma succión, 70, y están seguidos, en vez de repetir el mismo valor de succión una y otra vez, se podría indicar de la siguiente forma:

- succion: 70(10), ...

Con esto habríamos indicado que la succión es igual en todos los experimentos, lo que es más fácil y cómodo que poner:

- succion: 70,70,70,70,70,70,70,70,70,70,...

Por último, para repetir un valor tantas veces como repeticiones falten por completar el número de experimentos, se utiliza el carácter especial: '*'. Por ejemplo, para 10 repeticiones iguales:

- numero_experimentos: 10

- succion: 70*

Con esto indicamos a la plataforma que repita el valor 10 veces. Si antes se tuviesen otros elementos, el último completará las repeticiones que falten. Por ejemplo, para 7 repeticiones:

- numero_experimentos: 7
- succion: 30,40,50,70*

Para este caso, habrá 7 capturas en total, donde las succiones serán de 30, 40, 50, y el resto de 70, que serán 4 en total.

Si se usa este carácter y después se indican otros valores, esos valores serán ignorados. Este carácter no puede usarse con **vector_apertura_valvulas** de la forma que se ha explicado.

Configuración de patrones complejos de repeticiones

Si en vez de un solo valor, quisiésemos repetir varios valores, se ha añadido otra notación que ayuda a repetir un patrón. Para poner patrones hay que utilizar '{}' y cada elemento debe ir separado del siguiente por ','. Por ejemplo, si tenemos 20 experimentos y las duraciones de los estímulos son: 5 muestras de 30s, 5 de 60s, 5 de 30s y 5 de 60s, en vez de poner:

- duracion_estimulo: 30,30,30,30,30,60,60,60,60,60,30,30,30,30,30,60,60,60,60,60

Se puede poner:

- duracion_estimulo: {30(5),60(5)}(2)

Para los gases, también se pueden utilizar patrones. La notación que hay que utilizar es como se ha explicado anteriormente. Un ejemplo de patrones para los gases es el siguiente:

- vector_apertura_valvulas: 3(5)-{3-4(2)-1}(2),3(5)-{{3-4(2)-1}(2)-2}(2)

Esto sería equivalente a poner lo siguiente:

- vector_apertura_valvulas: 3-3-3-3-3-3-4-4-1-3-4-4-1,3.2-3.2-3.2-3.2-3.2-3-4-4-1-3-4-4-1-2-3-4-4-1-3-4-4-1-2

Como puede verse, en los patrones se puede incluir otros patrones, introduciendo uno dentro de otro.

Para utilizarse el carácter especial de la repetición de las medidas con los gases, debe de utilizarse con las llaves, indicando que se quiere repetir esa expresión tantas veces como repeticiones de gases se quieran analizar. Un ejemplo sería el siguiente:

- numero_experimentos: 3
- vector_apertura_valvulas: {3(5)-{3-4(2)-1}(2)}*

Con este ejemplo indicamos que el patrón de apertura de válvulas: 3-3-3-3-3-3-4-4-1-3-4-4-1 se repita tres veces.

En el anexo G se podrán varios ejemplos de ficheros de ejecución de muestras.

Comentarios

En los scripts de ejecución que se pasan a **PyHuele** se pueden añadir comentarios. Para añadir comentarios simplemente hay que utilizar el carácter especial: // al inicio de este. Es importante poner el comentario en una linea aparte, ya que no funciona si se pone a la mitad de una linea. Un ejemplo sería:

- //Esto sería un comentario en el script.

3.3. Requisitos funcionales y no funcionales

A continuación se exponen los requisitos funcionales y no funcionales que los módulos anteriormente y **PyHuele** deben de cumplir.

3.3.1. Requisitos funcionales del protocolo

- **RF1** - El orden en que se indique los elementos no debe ser un problema a la hora de tratarlos.
- **RF2** - El protocolo debe tener elementos para facilitar la repetición de elementos en caso de ser necesario.

3.3.2. Requisitos funcionales del software

- **RF3** - El programa tiene que facilitar la lectura de propiedades de las ejecuciones de un fichero.
- **RF4** - El usuario debe de poder indicar al programa si desea representar los datos de las ejecuciones.
- **RF5** - El programa tiene que tratar los casos de error indicando el error si se sale.
- **RF6** - El usuario podrá usar **PyHuele** en cualquier SO, mientras disponga del entorno de ejecución y las herramientas necesarias instaladas.

3.3.3. Requisitos no funcionales

- **RNF1** - El protocolo tiene que ser fácil de utilizar, con un lenguaje simple que ayude a su entendimiento.
- **RNF2** - El código de PyHuele se hará lo más modular posible, para que si es necesario ampliarlo o modificarlo, se modifique lo menos posible la estructura de este.

3.4. Diagramas funcionales y casos de uso

El sistema está compuesto de un actor principal, el usuario, que realiza los experimentos usando el software de control de la plataforma.

El diagrama de casos de uso se muestra en la figura 3.1, y muestra las funcionalidad que puede realizar el usuario.

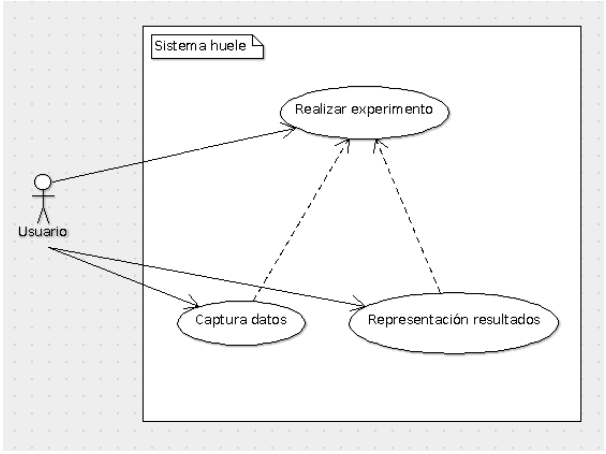


Figura 3.1: Diagrama de casos de uso de PyHuele

En la figura 3.2 se muestra la interacción de los distintos módulos entre sí, junto con las funciones que tiene cada módulo. El módulo de recogida y tratamiento de datos obtiene las características de la ejecución del script y trata los datos. Una vez que se han tratado los datos se pasan los que tienen que ver con los experimentos al módulo de captura de datos y los datos de representación al módulo de representación. El módulo de captura registra los odorantes y el módulo de representación una vez que se han capturado los datos representa todos los datos. Los códigos de los módulos se muestran en el anexo J.

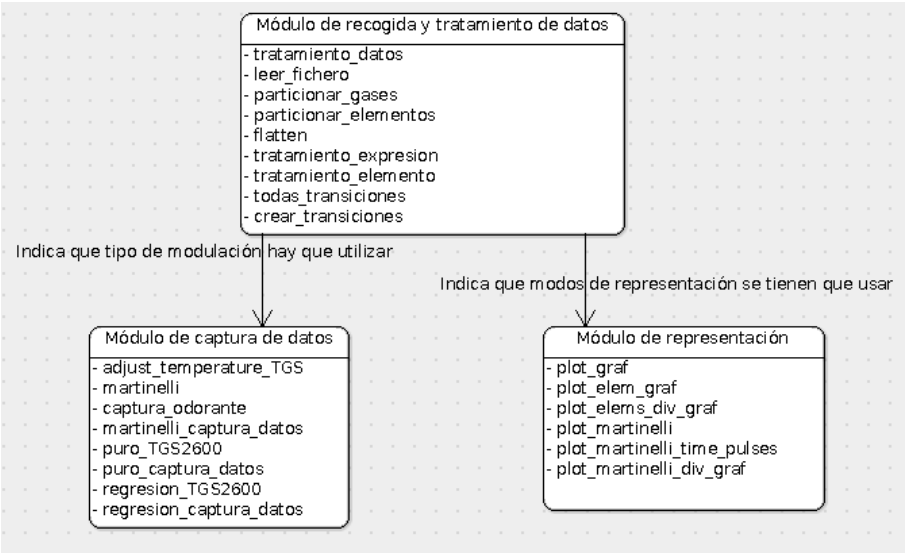


Figura 3.2: Iteracción de los distintos módulos de PyHuele

4

Desarrollo

En este capitulo se detalla todo el desarrollo de los distintos módulos y se procede a explicar como se han implementado. En el anexo I se muestran los cambios realizados al código que ya estaba implementado y en el anexo J se muestra todo el código desarrollado.

4.1. Tecnologías usadas para el pseudo-lenguaje de experimentación

Para el desarrollo del software de los distintos módulos se ha tenido que tener en cuenta los distintos elementos que forman la plataforma. Para el desarrollo del código se ha utilizado el lenguaje Python, que es un lenguaje interpretado y muy potente, con muchas librerías ya implementadas. Como interprete de Python se ha escogido el incluido por defecto en Debian, Python. Los sensores que se utilizan son de la empresa Figaro, el modelo TGS 2600(Figaro, 2016b).

La placa controladora de la plataforma es una BBB (Instrument, 2013), con un sistema Debian 7.8. Para el uso de sus múltiples pines de la BBB se ha utilizado un librería que trae implementadas distintas formas de activación de los pines. En los anexos se explica como poner a punto la BBB: en el anexo A se explica como instalar un SO a la BBB, en el anexo B se explica como configurar la BBB, en el anexo C se indican los pasos para crear una llave ssh, en el anexo D se muestran los valores máximos de las entradas de los pines de la BBB para no dañarla y en el anexo E se indica como ejecutar los programas en remoto y como hacer para que no se cierren cuando se corte la conexión.

4.2. Módulo de recogida y tratamiento de datos

En este módulo se obtienen los datos necesarios para realizar las ejecuciones, se tratan y se comprueba que no tengan errores. Si el script de ejecución tiene errores, el software de control finalizaría la ejecución indicando si el error está en la palabra clave, porque está mal escrita o son los parámetros de la palabra clave los que están mal, indicando que la expresión es errónea.

4.2.1. Librerías utilizadas

Para que el módulo de tratamiento de datos funcionase ha sido necesario utilizar unas librerías externas, para poder procesar la información del script:

- **re**[1]: Librería de expresiones regulares, que ayuda a encontrar cadenas y a partir expresiones entre otros.
- **numpy**[2]: Librería matemática open source que agrega un mayor soporte a vectores, matrices, transformadas de Fourier y otras funciones matemáticas a alto nivel.
- **datetime**[3]: Librería que implementa el tipo datetime, que maneja el tiempo en forma de dia, mes y año.
- **os**[4]: Conjunto de herramientas y rutinas del sistema operativo.
- **sys**[5]: Provee acceso a elementos que usa o mantiene el interprete de Python y a funciones que interactúan con el interprete.

4.2.2. Funciones y código desarrollado

Para el desarrollo del lenguaje de experimentación se ha elegido la sintaxis expuesta en la sección 3.2.1, porque además de ser fácil de usar, también es fácil de tratar a la hora de extraer la información. Con la ayuda de la librería de Python re, se divide la expresión por el elemento ':', y separamos la palabra clave del valor que tiene asignado.

Una vez que se ejecuta el software y se le indica la localización del script, **PyHuele** empieza a leer el fichero linea por linea usando la siguiente función:

```
def leer_fichero(path)
```

Una vez que se leen todos los datos, se procede a tratarlos, para ello se utilizan las siguientes funciones:

```
def tratamiento_datos(data)
def particionar_elementos(element,rep)
def particionar_gases(gases,rep)
def tratamiento_elemento(elem,rep)
def tratamiento_expresion(expr,cont,rep=0)
```

La primera función, tratamiento_datos(data), recibe todas las características del experimento como un conjunto de listas, donde cada lista está compuesta por la palabra clave y la expresión a tratar. Lo primero que se hace es comprobar que las palabras claves estén bien escritas y repetir las características asociadas a las palabras clave para todas las capturas. Por último se guardan los gases que se van a analizar.

Una vez que se tienen los parámetros de la palabra clave, se utilizan las funciones particionar_elementos o particionar_gases. Estas funciones dividen las expresiones por el delimitador ',' y para el caso de los odorantes por el delimitador '-' también.

[1] https://docs.python.org/2/library/re.html
[2] http://www.numpy.org/
[3] https://docs.python.org/2/library/datetime.html
[4] https://docs.python.org/2/library/os.html
[5] https://docs.python.org/2/library/sys.html

La función tratamiento_expresion recibe los elementos que han separado las funciones anteriores y busca patrones. Si los encuentra, va tratando cada elemento de este por separado y luego los junta. Para ello busca los '{', y hasta que no encuentra el final del patrón, '}', sigue tratando y juntando los elementos del patrón. Si no encuentra ningún patrón, se analiza los elementos por separado. Para analizar los elementos se llama a la función tratamiento_elemento(elem,rep).

El resultado de la función particionar_elementos debe guardarse en una sola lista y para ello se utiliza la función flatten, que retorna el parámetro que recibe en un sola lista.

Si en algún momento se encuentra un dato erróneo, se corta la ejecución con un mensaje de error.

4.3. Módulo de captura de datos

Este módulo utiliza elementos del código ya desarrollado, pero con varios cambios para adaptarlos al nuevo código de **PyHuele**. Antes eran tres programas diferentes, en los cuales sólo se podía realizar una captura y para poner otra nueva, se tenía que esperar a que acabase la anterior. Estos códigos se ha mejorado e integrado en el programa principal.

4.3.1. Ficheros que componen el módulo

Este módulo está compuesto de los siguientes ficheros:

- puro.py
- regresion.py
- martinelli.py
- EN_codigo_comun.py

Los tres primeros ficheros contienen los códigos de captura específico de cada técnica. El otro fichero restante, EN_codigo_comun.py, contiene el código que es común al resto de los ficheros, y que por limpieza y mantenimiento del código se ha decidido poner en otro fichero aparte.

4.3.2. Librerías utilizadas

Para que el código de captura de datos funcionase correctamente ha sido necesario utilizar unas librerías que controlan los pines de la BBB:

- **Adafruit_BBIO**[6]: Esta librería ha sido desarrollada por la empresa Adafruit. Esta librería sirve para poder utilizar los diferentes pines de la BBB. Tiene varias sub-librerías para cada conjunto de pines que vayan a utilizarse. De esta librería se utilizan las siguientes sub-librerías:
  - ADC - Uso de pines analógicos.
  - GPIO - Uso de los pines generales de entrada/salida.
  - PWM - Uso de los pines que generan un pulso con modulación.
  - DHT - Permite el control de los sensores de temperatura y humedad.

[6] https://learn.adafruit.com/setting-up-io-python-library-on-beaglebone-black/using-the-bbio-library

- **time**[7]: Esta librería provee diversas funciones para la medición del tiempo.
- **datetime**[8]: Librería que implementa el tipo datetime, que maneja el tiempo en forma de dia, mes y año.
- **threading**[9]: Librería que emula el modelo de control de hilos de Java.
- **os**[10]: Conjunto de herramientas y rutinas del sistema operativo.

4.3.3. Funciones y código desarrollado

Para este módulo se han adaptado las funciones de captura de datos, que son llamadas desde el software de control. Se tienen 3 funciones diferentes, una para cada modo de captación de datos y según el tipo escogido se les pasa unos parámetros u otros.

```
def puro_captura_datos(datetime,data)
def regresion_captura_datos(datetime,data)
def martinelli_captura_datos(datetime,data)
```

Para la captura de datos con el algoritmo de temperatura constante y el algoritmo de temperatura variable con codificación en amplitud, los datos se captan de forma síncrona y cada segundo se recogen datos del sensor y se almacenan. En cambio, la modulación propuesta por Martinelli es asíncrona, ya que depende de la velocidad a la que se realicen los pulsos.

La captura con temperatura constante, como se explicó en la sección 2.6.1, no aplica ningún cambio a la señal, sino que tras obtener los datos del sensor, usando la función ADC.read_raw(), los guarda en el fichero pertinente. Para la modulación con codificación en amplitud, se leen los datos usando la misma función y se realiza una regresión a los datos obtenidos, para así obtener la nueva temperatura de calentamiento del sensor.

Como se explicó en la sección 3.1.2, estas técnicas son síncronas, registrando cada segundo un valor del sensor. Para ello se mide el tiempo de captura de la siguiente forma:

```
time_ini = time.time()
#Se realiza la captura de los datos del sensor
time_end = time.time()
time.sleep(ENC.SLEEP - (time_end - time_ini))
```

Se registra el instante de tiempo antes de capturar los datos. Tras haberlos capturado, se registra otra vez el tiempo y se restan los valores, obteniendo el tiempo de operación. La diferencia entre el periodo de muestro y el tiempo de operación marca la espera hasta la siguiente captura. De esta forma se consigue captar una muestra por segundo (periodo de muestro), obteniendo un sistema de medida en "tiempo real". Esto se muestra en la figura 4.1.

Para el algoritmo de temperatura variable y codificación en frecuencia, como hay que esperar al cambio entre el estado bajo y el estado alto, se mide el tiempo de cada uno de los estados, teniendo así la medida de una transición entre estados. Esta forma de capturar los datos es asíncrona, ya que no se pueden captar las muestras en un tiempo fijo, debido al efecto de los gases en el sensor.

[7] https://docs.python.org/2/library/time.html
[8] https://docs.python.org/2/library/datetime.html
[9] https://docs.python.org/2/library/threading.html
[10] https://docs.python.org/2/library/os.html

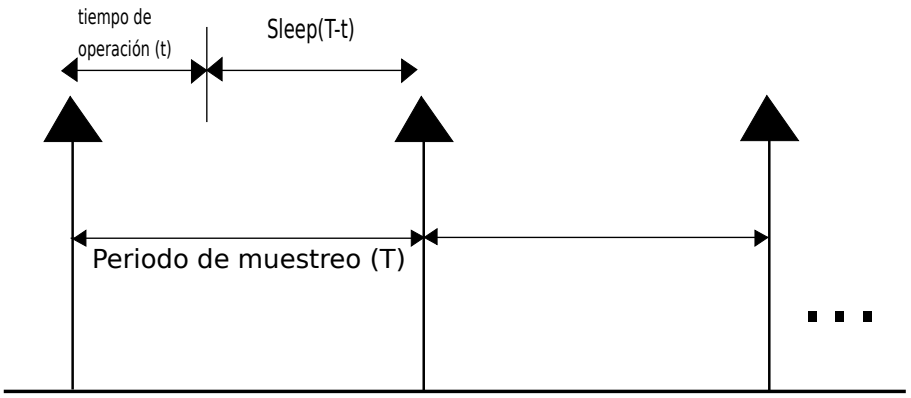


Figura 4.1: Esquema de captura de datos. El valor de t es el tiempo que ha estado captando datos y T es el tiempo de muestro.

```
#Espera pulso de caida
instante_captura_ini=datetime.now()
value_down = GPIO.wait_for_edge(sensorPin555, GPIO.FALLING)
ini_pulso = time.time()

value_up = GPIO.wait_for_edge(sensorPin555, GPIO.RISING)
ini_up = time.time()

value_down = GPIO.wait_for_edge(sensorPin555, GPIO.FALLING)
#Espera pulso de bajada
fin_pulso = time.time()

time_down = ini_up - ini_pulso
time_up = fin_pulso - ini_up
#Duracion del pulso en estado alto
time_pulso = fin_pulso - ini_pulso
#Duracion del pulso completo
```

En este modo de captura se han implementado dos formas de captar los odorantes. La primera consiste en tener un tiempo constante, una ventana de tiempo fija, y durante ese tiempo captar el máximo de pulsos posibles. La segunda forma, consiste en fijar un número de transiciones y esperar a que el sistema las complete y ver el tiempo que tarda en realizarlos.

Este código es el que se usa para obtener el tiempo o el número de pulsos. Según el modo de ejecución, se devuelve el tiempo que se ha tardado en realizar una transición completa o un 1 indicando que se ha realizado una transición entre estados.

```
    time_martinelli = martinelli(...)
    instante_captura = datetime.now()
    #Codigo de escritura en ficheros

    if mode_execution == 1:
        return time_martinelli
    else:
        return 1
```

Para este tipo de captura de datos, también se han implementado dos formas de variar

la temperatura, una donde la temperatura cambia del mínimo al máximo instantáneamente y viceversa, y otra forma donde se aumenta o se disminuye la temperatura paulatinamente. La primera forma cambia la temperatura al realizarse una transición entre dos estados, y la segunda forma varía la temperatura con cada pulso.

Para las funciones comunes como la apertura de ficheros, la captura de datos de temperatura y humedad, la apertura de las válvulas o el cierre de los descriptores de los ficheros, se ha creado una librería nueva que se ha llamado: EN_codigo_comun. Estas funciones comunes son:

```
def abrir_electrovalvulas(electrovalvulas)
def cerrar_electrovalvulas()
def imprimir_electrovalvulas(elecs)
def create_files(nameFile,path,folder,samplesinicio,ahora)
def open_files(ruta_fichero,ruta_fichero_data,ruta_fichero_tyh)
def file_TGS2600(vec_open_valve,succion,heat2600,
    tendencia,heatPin2600,tiempo,switch,algoritmo,ruta,nameFile,
    nameFile_data1,nameFile_data2,nameFile_tyh,f,opcion,tespera,
    samplesinicio)
def measure_tyh(h)
def cierre(f,g,h,t)
```

Medición de Temperatura y Humedad

Un aspecto común de todos los tipos de captación de odorantes, es la forma en la que se registran los valores de humedad y de temperatura. Para ello se ha expedido un hilo aparte, que capta los datos de la temperatura y humedad cada minuto. Este hilo ejecuta un bucle que se repite hasta que se le indica que pare.

El hilo se expide al principio de la ejecución, y se sincroniza con el hilo principal, para que capte datos cuando se empiecen a registrar datos del sensor, o cuando se va a finalizar la ejecución del software de control y se quiere finalizar su ejecución.

Para captar las muestras cada minuto se utiliza el mismo sistema de "tiempo real"que se utiliza en las técnicas de obtención de datos con temperatura constante y con temperatura variable y codificación en amplitud.

La función que se encarga de captar los datos de la temperatura y la humedad es:

```
def measure_tyh(h)
```

4.4. Módulo de representación de datos

Este módulo se encarga de representar los datos obtenidos. Se han implementado diferentes formas de representar los datos, según el tipo de modulación escogida. Estas funciones son nuevas y no se tenía código implementado, así que se parte desde cero.

4.4.1. Ficheros que componen el módulo

Este módulo está compuesto de un solo fichero,que contiene todas las funciones para representar los datos:

- plot.py

4.4.2. Librerías utilizadas

Para desarrollar el código de representación de datos se han utilizado las siguientes librerías:

- **matplotlib**[11]: Librería con funciones que ayudan a representar datos.
- **re**[12]: Librería de expresiones regulares, que ayuda a encontrar cadenas y a partir expresiones entre otros.
- **os**[13]: Conjunto de herramientas y rutinas del sistema operativo.

4.4.3. Funciones y código desarrollado

Cada función que representa datos recibe como parámetro un diccionario de Python, que contiene como clave las características del experimento en forma de tupla:

- El tiempo de análisis de cada gas.
- Las muestras iniciales.
- El tiempo entre muestras.
- El tipo de ejecución de martinelli.
- El modo de ejecución de martinelli.
- La temperatura mínima para martinelli.
- La temperatura máxima de martinelli.
- La carpeta donde se va han guardado los datos.

Las funciones también reciben el instante de tiempo en que se inició la ejecución de **PyHuele** y la ruta donde se van a guardar las representaciones. Las gráficas se guardan en la misma carpeta donde se encuentran los datos que se van a representar, en otra carpeta nueva llamada: images.

Un ejemplo de como estaría compuesto el diccionario es el siguiente:

```
{(tiempo de analisis de un gas,muestras iniciales, tiempo entre
gases, el número de gases analizados, forma de
cambiar la temperatura en  Martinelli, modo de
ejecución en martinelli, Tmin,Tmax,folder):
[file_name1, file_name2, ..., file_nameN]}
```

Cada vez que se llama a una función que representa los datos, se obtienen todos los datos de la clave del diccionario, y dependiendo del tipo de ejecución se usan unos datos u otros. Esto se ha desarrollado así para que el código sea lo más uniforme posible.

Para el modo de temperatura fija y el de temperatura variable y codificación en amplitud se han desarrollado las siguientes funciones de representación:

[11] http://matplotlib.org/
[12] https://docs.python.org/2/library/re.html
[13] https://docs.python.org/2/library/os.html

```
def plot_graf(dictionary,ahora,path)
def plot_elem_grafs(dictionary,ahora,path)
def plot_elems_div_graf(dictionary,ahora,path)
```

Con cada una de estas funciones se imprime unos datos diferentes. La primera, plot_graf, realiza 4 gráficas de un mismo archivo de datos. Imprime la resistencia interna, el valor en voltios captado por el sensor y la temperatura de este en una gráficas separadas, y por último imprime los 3 valores anteriormente mencionados en una misma.

La función plot_elem_grafs, crea 3 gráficas diferentes para un conjunto de archivos. En cada gráfica se representa un mismo valor, resistencia del sensor, el valor captado o la temperatura, para todo el conjunto de datos y poder comparar los resultados obtenidos.

La última función, plot_elems_div_graf, es como la anterior, pero con la diferencia de que imprime cada transición en un panel, para analizar de forma más minuciosa cada transición.

Para representar todos los datos, se han utilizado dos funciones auxiliares, que son las siguientes:

```
def plot_vertical_lines(s_ini,switc,n_times,CTE)
def normalizar(x)
```

La primera imprime las lineas verticales de separación entre los distintos gases, y la segunda normaliza un conjunto de datos que se le pase, poniéndolo entre 0 y 1. Esta última se utiliza para imprimir todos los valores juntos.

Para representar los datos captados por el algoritmo de temperatura fija y codificación en frecuencia, no se han utilizado las mismas funciones, sino que se han desarrollado otras nuevas, ya que como se ha dicho antes, este modo es asíncrono. Para ello se tienen las siguientes funciones:

```
def plot_martinelli(dictionary,ahora,path)
def plot_martinelli_div_graf(dictionary,ahora,path)
```

La primera función imprime las transiciones de los estados y la temperatura. La temperatura se imprimirá como una onda cuadrada que coincide con las transiciones de los estados, si se eligió que cambiase de forma inmediata, o se imprimirá la huella que se obtiene al variar la temperatura de forma síncrona con los pulsos.

La última función, plot_martinelli_div_graf, imprime los valores de las transiciones de los odorantes en diferentes paneles en una misma gráfica, para analizar las transiciones de forma más minuciosa.

Para representar estos valores, también se han utilizado unas funciones auxiliares:

```
def martinelli_vertical_separation(time_total,
    mexecution,switch,CTE,sini,n_times)
def obtener_tiempo_pulsos_martinelli(time_total,
    switch,CTE,s_ini,ntran)
def martinelli_pulses(time_up,time_down,ini,fin)
def martinelli_temp_pulses(time_total,
    temp,texecution,ini,Tmin,Tmax)
def martinelli_ventana_tiempo(time_total,switch,
    CTE,sini,Tmin,Tmax)
def martinelli_numero_pulsos(time_total,argtemp,
    switch,CTE,sini)
```

```
def martinelli_temp_pulses_div_graf(time_total,
    argtemp,switch,mexecution,CTE,sini,Tmin,Tmax)
```

Estas funciones tienen como propósito dibujar las lineas de separación vertical entre los gases, martinelli_vertical_separation; obtener el tiempo de ejecución de las transiciones, obtener_tiempo_pulsos_martinelli; obtener el tiempo de los pulsos, martinelli_pulses y obtener la temperatura en función del tiempo, martinelli_temp_pulses. Las funciones martinelli_ventana_tiempo y martinelli_numero_pulsos obtienen las transiciones entre estados que se van a representar en los distintos paneles de las gráficas.

Para leer los archivos con los datos se ha desarrollado la siguiente función, que lee del archivo las columnas solicitadas:

```
def get_data_columns(name,columns)
```

A esta función se le pasa el nombre del archivo que desea leerse y que columnas leer. Para indicar que se quiere leer la primera columna se pone un 0, así hasta la última columna, la N, que se indica con N-1. El resultado es una lista de listas, donde cada sub-lista es una columna leída; los valores devueltos son floats.

5

Experimentos Realizados y Resultados

En este capítulo se realizan pruebas al software de control y al pseudo-lenguaje de experimentación con diferentes experimentos, para comprobar que ambos funcionan correctamente.

5.1. Experimentos realizados

Los siguientes experimentos se han realizado para comprobar que los códigos desarrollados funcionan correctamente y que el pseudo-lenguaje implementado sea el correcto, pudiendo usar los algoritmos de captación sin problemas.

Se han realizado varios experimentos para las diferentes técnicas de captación de odorantes implementadas, observando la reproducibilidad de la plataforma y la huella dejada por los odorantes. Para realizar los siguientes experimentos se han utilizado varias concentraciones, comprobando la reacción del sensor a diferentes odorantes con algunas de las concentraciones. Los códigos de las concentraciones utilizadas en las figuras de este capítulo se muestran en el anexo F, donde también se muestra los instrumentos necesarios para crearlas.

Para ver si se obtienen valores reproducibles con el pseudo-lenguaje desarrollado y el software de control, se han realizado varias capturas seguidas con las mismas secuencias de odorantes y las mismas concentraciones. Si los datos que se obtienen son reproducibles, para las diferentes capturas se deberían de obtener valores aproximados de temperatura, voltaje y resistividad. Hay ciertos parámetros que en determinadas condiciones pueden alterar las medidas tomadas como: la temperatura del ambiente, el drift del sensor que consiste en obtener distintos valores en las mismas condiciones debido a la memoria del sensor o el tiempo que el sensor ha estado usándose, para este tipo de sensores el fabricante recomienda que antes de realizar mediciones válidas el sensor este una semana captando datos (Figaro, 2016b).

En esta sección se muestran los resultados y gráficas más relevantes de los múltiples experimentos realizados con la plataforma. En el anexo H se muestran más resultados y gráficas complementarias.

5.1.1. Captura de datos mediante temperatura constante

En los experimentos realizados con este método de captación de odorantes se ha captado primero la huella que dejan los gases, para ver si estas son muy parejas o diferentes y tratar ver si estos gases serían fácilmente reconocibles por un clasificador o incluso a ojo.

Para ello se han realizado 3 capturas diferentes, con 90 muestras iniciales antes de analizar los gases, y analizando cada gas 90 segundos dejando otros 90 segundos de aire entre los odorantes. La combinación de gases es: muestras iniciales(aire), aire, etanol, aire, butanol, aire, metanol y aire. Los resultados se muestran en la figura 5.1 y el script del pseudo-lenguaje usado para la ejecución del experimento se encuentra en G.1.2.

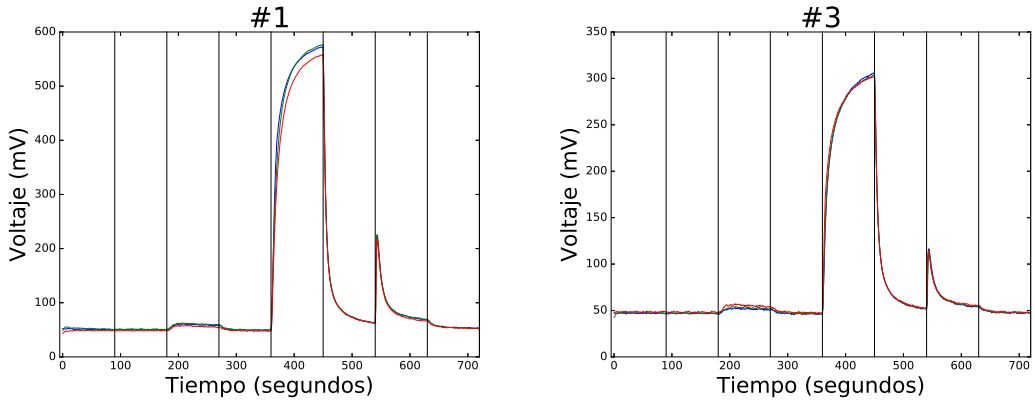


Figura 5.1: Efecto de los gases en el sensor usando el algoritmo de toma de datos sin modulación. La figura de la izquierda usa la concentración #1 y la imagen de la derecha usa la concentración #3. Los gases representados son: muestras iniciales, aire, etanol, aire, butanol, aire, metanol y aire, en ese mismo orden. En las figuras, las lineas verticales separan los gases. El script del pseudo-lenguaje usado para la ejecución de esta captura es G.1.2

Como puede verse en la figura 5.1 el etanol es casi imperceptible, mientras que el metanol y el butanol son los que más fácilmente reconoce el sensor. El butanol es el odorante del que más señal se obtiene y el metanol da un pico, satura y luego se estabiliza. Cada figura tiene 3 capturas diferentes para poder compararlas unas con otras, donde cada color representa una captura. Como puede verse, la huella dejada por estos gases es muy característica, siendo sencillo distinguirlos entre sí. La diferencia en la intensidad de la señal entre ambas figuras se debe a que se han usado concentraciones distintas.

Para probar la reproducibilidad de la plataforma y la validez del pseudo-código desarrollado, se han realizado varias capturas; los resultados de algunas de ellas se pueden ver en la figura 5.2.

Como se puede ver se obtienen unos datos muy reproducibles, con lo que para este algoritmo se obtienen resultados satisfactorios. Se puede ver que los datos tienen una pequeña variación, en la figura de la izquierda, de 60 mV aproximadamente, que puede deberse a la temperatura del ambiente o a que el sensor tuviese una temperatura mayor debido al uso y de 5mV, aproximadamente en la figura de la derecha, donde la concentración de odorantes es menor. Se han realizado 10 capturas diferentes, que se representan juntas y cada captura se representa por un color diferente.

En el pie de la figura pueden verse todas las transiciones entre odorantes, representadas como lineas verticales. En el resto de figuras de este capítulo el esquema usado es el mismo. El script del pseudo-lenguaje utilizado para realizar las capturas se muestra en G.1.2. Los valores de la resistencia que se muestran en estas figuras, como en el resto de figuras se calculan siguiendo la fórmula que indica el fabricante (Figaro, 2016b).

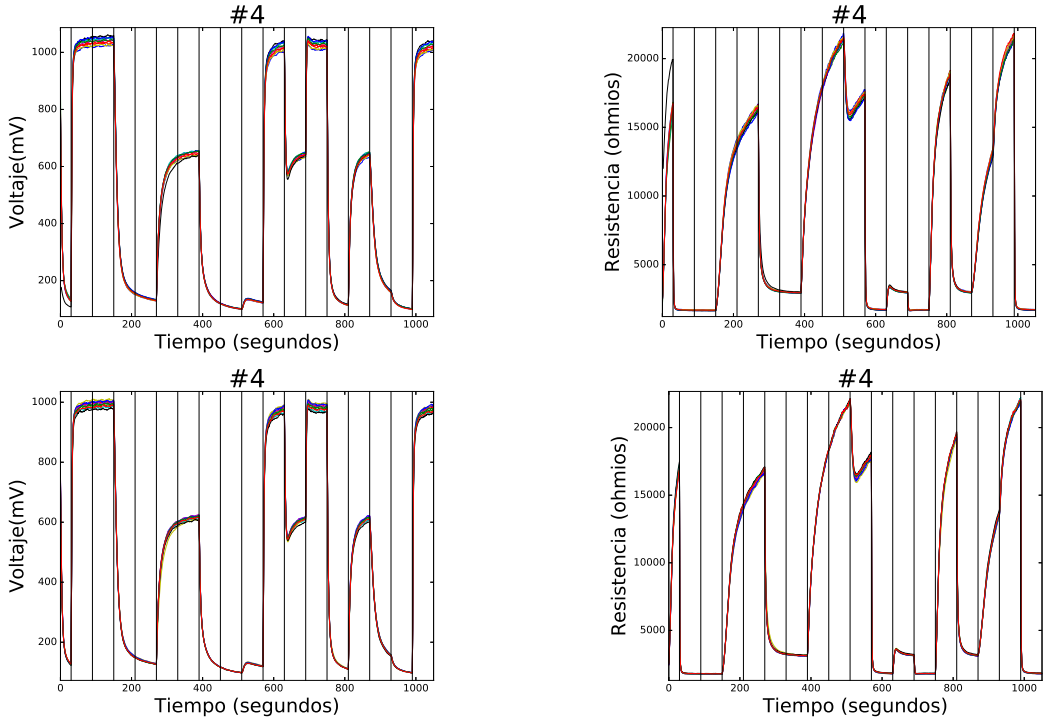


Figura 5.2: Capturas de datos con el algoritmo de temperatura constante. Se han capturado todas las transiciones posibles. La concentración utilizada es #4. El orden de los odorantes es: muestras iniciales, metanol, metanol, etanol, etanol, butanol, butanol, aire, aire, etanol, metanol, butanol ,metanol, aire, butanol, etanol, aire y metanol. Cada gas se analiza 60 segundos todos seguidos. El script del pseudo-lenguaje utilizado se muestra en G.1.2

En el anexo H.1 se muestran más representaciones usando este modo de captación de datos.

Como los experimentos realizados con este método de captura queda demostrado que el pseudo-lenguaje desarrollado permite explorar el funcionamiento de la plataforma variando elementos simples como el tiempo de captura de un odorante o el nombre del fichero donde guardarlo entre otros.

5.1.2. Modo de captura de datos mediante temperatura variable y codificación en amplitud

El algoritmo de temperatura constante y modulación en amplitud se auto-regula según va captando datos, realizando para ello un regresión de los datos del odorante ya registrados, evitando que la señal llegue al límite de su valor y se sature. El número de muestras iniciales, o ventana de tendencia, determina cuanto oscila la señal de la temperatura. Esto se muestra en la figura 5.3, y se debe a que si la ventana de tendencia es pequeña el algoritmo tiene menos datos en los que basarse y la señal oscila más, mientras que cuanto mayor sea la ventana más datos tiene para la regresión y menos oscila la señal.

En estas representaciones, se ha utilizado la muestra #0, y se han representado todas las transiciones posibles. El script del pseudo-lenguaje usado para este tipo de captura se encuentra en G.1.3.

Como se puede ver en la figura 5.3, la señal de la temperatura oscila menos cuanto mayor sea la ventana de tendencia, esto junto al valor captado por el sensor y la resistencia ayudan a diferenciar los odorantes. Un parámetro clave en esta forma de captar los datos es la ventana de

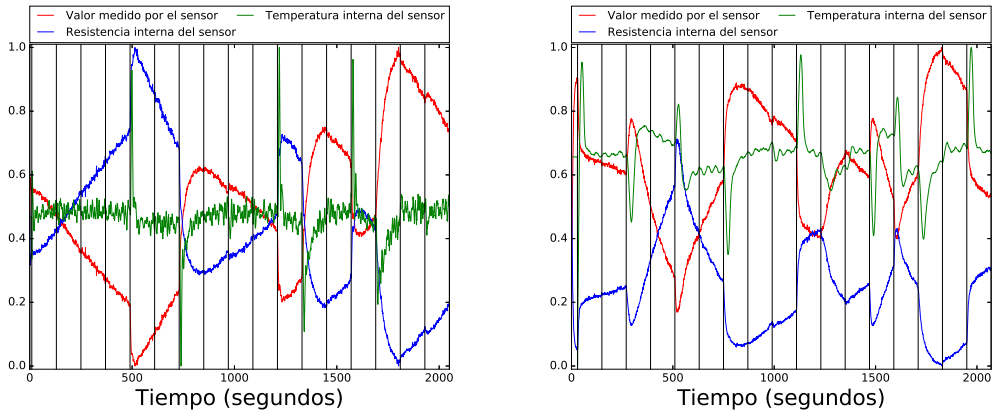


Figura 5.3: Captura para ver la señal de la temperatura con diferentes muestras iniciales. La figura de la izquierda tiene 10 muestras iniciales y la de la derecha 30 muestras iniciales. La concentración usada es #0. El script con el pseudo-lenguje usado para realizar estas capturas es G.1.3

tendencia. La diferencia de tamaño en la ventana de tendencia puede dar resultados diferentes a la hora de obtener los datos, como puede verse en las primeras transiciones en ambas figuras. En la figura de la derecha, en la tercera transición, el valor del sensor aumenta repentinamente mientras que en la izquierda el valor disminuye constantemente. Este fenómeno puede deberse al tamaño de la ventana de tendencia y se estudiará en un Trabajo de Fin de Máster, que se realizará el año que viene, para buscar su valor óptimo, ya que puede ser clave para la distinción y clasificación de distintos odorantes.

Para ver la reproducibilidad de este tipo de algoritmo, se han realizado varias capturas que se pueden ver en la figura 5.4. Como se observa, salen datos reproducibles y las variaciones que se obtienen son del orden de 5 mV, siendo estas menores que las obtenidas mediante el algoritmo de temperatura constante, por lo que los resultados son satisfactorios. Estas gráficas muestran un total de 10 capturas diferentes, donde cada captura es representada con un color, captando todas las transiciones de los diferentes gases.

En el anexo H.2 se muestran más capturas usando este método de captación de odorantes.

Con estas gráficas se muestra que el pseudo-lenguaje funciona bien con este método, pudiendo variar los parámetros sin esfuerzo, realizando búsquedas paramétricas de los mejores valores para la captura de los odorantes. Esto se muestra en la figura 5.4, donde la variación de la ventana de tendencia hace oscilar menos la señal de temperatura variando la forma de registrar los datos buscando el mejor valor para la ventana de tendencia.

5.1.3. Modo de captura de datos mediante temperatura variable y codificación en frecuencia

Con el algoritmo propuesto por Martinelli se han realizado experimentos tratando de reproducir los experimentos publicados (Martinelli et al., 2012). Se han implementado dos formas diferentes de captar los odorantes, la primera es midiendo con un tiempo constante y la segunda con un número de pulsos constante. La temperatura del sensor también se puede variar de dos formas: de forma instantánea o de forma progresiva. Con el pseudo-lenguaje desarrollado se puede variar entre las diferentes formas de registrar los datos de forma simple, sin necesidad de tener que modificar el código como se tenía que hacer antes, simplicando la captura de los datos. También esto es aplicable a la forma de variar la temperatura, ya que antes no se podía elegir que modelo de temperatura usar y ahora se puede usar cualquiera indicándolo en el script.

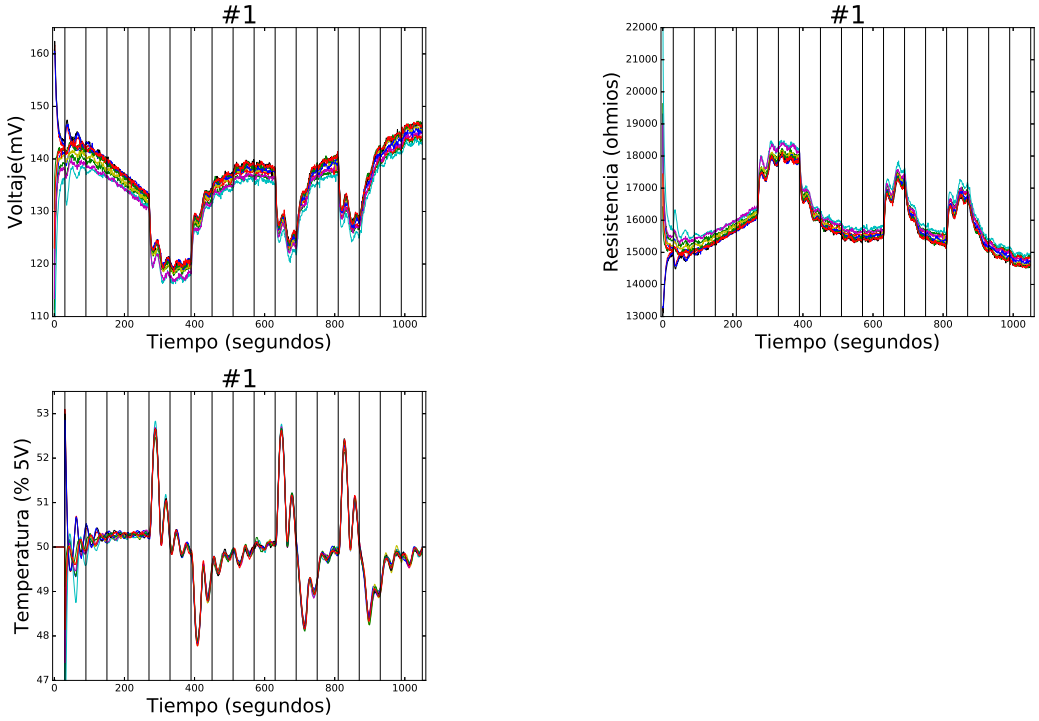


Figura 5.4: Capturas de datos con el algoritmo de temperatura variable y codificación en amplitud. La concentración es #1, con 60 segundos para cada odorante y el script con el pseudolenguaje se encuentra en G.1.3

Tiempo constante

Para el método de tiempo constante se deja un tiempo fijo durante el cual se captura el odorante. Este método no llega a completar pulsos completos, ya que el tiempo limita la captura de los gases. Para codificar los datos que se obtienen se utilizan el número de transiciones que se registran, junto con los estados alto o bajo que se llegan a completar y la huella que dejan.

Usando el método de la captura por tiempo, se obtienen las figura 5.5. En ellas se ve como algunos pulsos iniciados no llegan a completarse y se quedan a medias en el estado alto o en el bajo. Las figuras representan 10 capturas de las transiciones del metanol y del etanol, donde cada color representa una captura diferente. La ventana de tiempo que se ha empleado es de 120 segundos.

Como se puede ver en los paneles de la figura 5.5, las transiciones entre estados se quedan sin completarse y en las distintas capturas puede verse como los estados, tanto bajo o alto, se cortan. En cada figura puede verse la media y la varianza de las transiciones entre estados y de los estados incompletos. La primera fila representa la media y la varianza de las transiciones completas entre estados, la segunda la media y varianza de los estados bajos incompletos y la última de los estados altos.

Como puede observarse en la figura 5.5, una ventana de 120 segundos no es suficiente para discriminar los distintos odorantes utilizados en el experimento, ya que los valores de las medias y varianzas son muy similares.

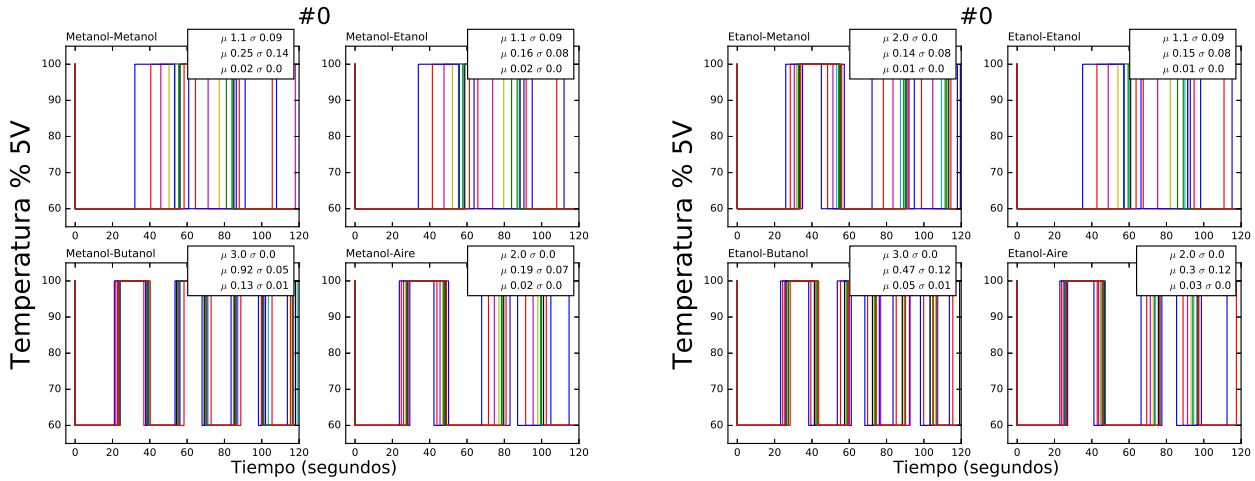


Figura 5.5: Transiciones de los diferentes gases. A la izquierda las transiciones del metanol, a la derecha del etanol. Todas realizadas con concentración #0.

Pulsos constantes

Con este método se deja un número de transiciones fijas y se mide el tiempo que tarda el sistema en completarlas. Según el tipo de odorante se consiguen unos tiempos diferentes al resto, dependiendo de como cambie el odorante la resistencia del sensor, haciendo que el circuito oscile más o menos. Para este tipo de capturas, se ha hecho que la temperatura vaya variando gradualmente con los pulsos, para ver si la huella que dejan los odorantes es distinguible.

La forma de codificar los odorantes con este modo de captación es midiendo el tiempo que se tarda en completar las transiciones requeridas, junto con la huella que se obtiene con el cambio de temperatura.

En la gráfica 5.6, se muestran diferentes capturas con distintas muestras de odorantes para una transición entre estados. En ellas se puede ver las huellas de los gases y la variación en la amplitud de estas. En cada panel se muestra el tiempo que se ha tardado en obtener 10 transiciones entre estados.

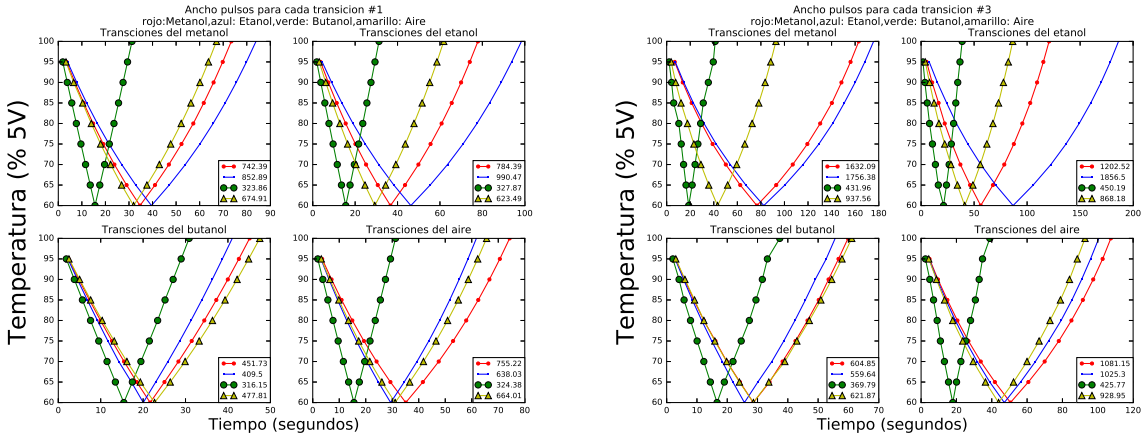


Figura 5.6: Transiciones de los diferentes odorantes. El panel de la izquierda representa la huella dejada por los odorantes con la concentración #1 y el de la derecha con la concentración #3. En cada sub-panel se indica el tiempo que tarda en realizar las 10 transiciones.

En cada panel se muestran las transiciones de un odorante hacia el resto. El título de cada

figura indica el odorante inicial, y el color el gas de destino.

La figura 5.7 muestra el tiempo que se tarda en obtener 10 transiciones entre estados para todas las transiciones de odorantes de 5 capturas diferentes para todas las concentraciones. Se ha realizado la media de las transiciones entre estados de todas las capturas y se han representado en un histograma. Cada panel es el histograma del tiempo que se tarda en realizar las 10 transiciones entre estados. En el eje x se representa el tiempo, en periodos de tiempo, y en el eje y se representa el número de eventos.

En el histograma se pueden ver una reproducibilidad entre las distintas capturas, ya que el número de eventos es el casi el mismo para las diferentes muestras, aunque estén separadas debido a la diferente concentración de los gases. Las capturas de la izquierda son las que menos tardan, mientras que las que se encuentran más a la derecha son las que más tardan. Los valores que se encuentran a la izquierda son los de las concentraciones cuyos odorantes tienen una disolución mayor, que va disminuyendo en las concentraciones de los odorantes de la derecha.

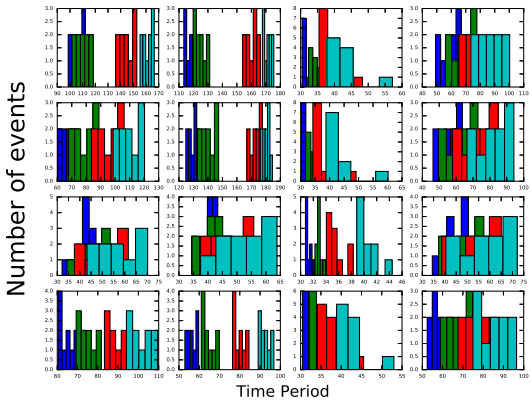


Figura 5.7: Histograma de capturas de los pulsos para distintas muestras. Los colores representan las diferentes concetraciones usadas: el azul oscuro representa la concentración #0, el verde representa la concentración #1, el rojo representa la concentración #2 y el azul claro representa la concentración #3

En el anexo H.4 se muestra un histograma de los tiempos separados para las diferentes concentraciones.

Cada panel representa una transición entre los diferentes estados de la siguiente forma: en las filas, las transiciones son metanol, etanol, butanol y aire, y en las columnas las transiciones son también metanol, etanol, butanol y aire.

Con los experimentos realizados con este método de captura se ha demostrado la facilidad en el uso de los diferentes modos de codificación en frecuencia usando el pseudo-código desarrollado. En los códigos iniciales, para variar entre los modos de captura se tenía que cambiar el código del programa, porque no se tenían ambos modos al mismo tiempo; por lo que el pseudo-lenguaje facilita mucho el uso de la plataforma.

Con los resultados obtenidos de las capturas realizadas con tiempo y pulsos constantes (figuras 5.5 y 5.6) se observa que con el modo de pulsos constantes se obtienen unos resultados más discriminantes, debido a la huella dejada por el gas y al tiempo que tarda en completar los pulsos. Sin embargo, estos resultados no son definitivos y deben estudiarse más a fondo, ya que con una ventana de tiempo mayor a la utilizada el resultado que se obtenga puede ser mejor que el obtenido.

Otro aspecto importante con este tipo de modulación son los 16 pulsos totales que se dividen

en dos grupos de 8 pulsos cada uno. Este valor se ha dejado igual al que propone Martinelli en su trabajo (Martinelli et al., 2012), pero se puede variar, buscando cual es el número de pulsos óptimo para la captura de los odorantes.

6

Problemas, conclusiones y trabajo futuro

6.1. Problemas encontrados

Durante el desarrollo de este TFG se ha tenido que lidiar con varios problemas con la plataforma. El primero de ellos apareció cuando el motor de succión se estropeó y hubo que sustituirlo por otro más moderno. Este problema fue la causa de que se introdujese en el pseudo-lenguaje la palabra clave: **reposo**, para que el sistema realizase esperas entre experimentos y que el motor no se sobrecalentase en exceso.

También se tuvo que reinstalar el SO de la BeagleBone Black, debido a un problema con las librerías que controlan los pines PWM, ya que no respondían bien. Esto mantuvo la plataforma detenida durante un tiempo, debido a que se creía que el problema residía en la BBB y no en el software. Con la reinstalación se solucionó el problema.

Existe otra incidencia no resuelta debido a un fallo en el software de la BBB, por el cual la memoria interna se encuentra al límite de su capacidad tras un tiempo de utilización de la plataforma, por cuestiones ajenas al código desarrollado. Esto ocasiona que, en estos casos, no se pueda acceder de forma remota y que no se puedan crear ficheros nuevos. Cabe la posibilidad de que sea provocado por un fichero temporal que almacena información del sistema.

6.2. Conclusiones y discusión

Con la realización de este TFG se han puesto en práctica varios conocimientos adquiridos a lo largo de la carrera. En la realización se han estudiado varios elementos de distintos ámbitos y se ha creado un software con un protocolo para el desarrollo de experimentos y la representación gráfica de estos.

Una vez leídos los trabajos previos y entendidos tanto la parte teórica, como la parte práctica, se comenzó la realización de la aplicación **PyHuele**, junto con el pseudo-lenguaje para la realización del protocolo. El pseudo-lenguaje es muy sencillo de utilizar y sirve para programar la realización de varios experimentos de forma simple. El software de control es muy sencillo y avisa al usuario cuando este comete un error al introducir los datos en el script de ejecución. La sencillez tanto en el pseudo-lenguaje como en el software de control es importante, porque así

cualquier persona que no conozca el funcionamiento interno de la plataforma puede usarla sin problemas. Además el software de control permite la representación de los datos captados para compararlos de forma visual.

Este pseudo-lenguaje es de gran ayuda para realizar experimentos de forma fácil, sin necesidad de estar modificando código, ya que antes se tenían que realizar varios cambios, ralentizando los experimentos y sin obtener el máximo partido de la plataforma. Se ha demostrado que con el pseudo-lenguaje es posible realizar búsquedas parámetricas en los diferentes modos de captación de odorantes, donde se utilizan varios elementos configurables. Por ejemplo la ventana de tendencia(en el modo temperatura variable y codificación en amplitud) o la duración de 8 pulsos para cada semi periodo(en el de frecuencia), para buscar parámetros óptimos. También se pueden definir patrones complejos en los script de ejecución, facilitando la realización de experimentos y evitando la repetición de elementos.

Con esto se puede obtener el mejor algoritmo de captación de odorantes de forma simple, ya que como se muestra en los experimentos realizados se observa que con el algoritmo de temperatura variable y codificación en amplitud se consigue una mayor reproducibilidad que con el de temperatura constante y esto puede ayudar a diferenciar mejor los odorantes que se analizan.

6.3. Trabajo futuro

A continuación se muestran diversas tareas en las que sería interesante profundizar en el Trabajo de Fin de Máster:

- Migración de los códigos Python2 a Python3: la versión 2 de este lenguaje de programación se encuentra en sus 3 últimos años de mantenimiento. Es recomendable realizar la migración por las ventajas de la nueva versión.
- Creación de una interfaz gráfica para facilitar la interacción con el programa.
- Captación de un gran volumen de datos para su clasificación usando para ello redes neuronales o algoritmos de machine learning, y comprobar si la plataforma obtiene datos discriminantes.
- La mejora de los elementos hardware, imprimiendo los circuitos de la protoboard en un circuito integrado.
- Implementación de un nuevo tipo de variación en la temperatura en el modo de temperatura variable y codificación por frecuencia, donde la temperatura varía cada segundo.
- La realización de una búsqueda paramétrica para encontrar los parámetros de captura óptimos. Por ejemplo ajustes en la ventana de tendencia, para la codificación en amplitud y el número de pulsos de reloj óptimo, junto con el número de transiciones de estado, en el algoritmo de temperatura variable y codificación en frecuencia, etc, para que los datos sean lo más discriminantes posibles.
- La preparación del lenguaje para que sea lo más extensible posible sin necesidad de alterar mucho el código, en el caso de que se añada un nuevo tipo de modo de operación.

Bibliografía

Española, R. R. A. (2010). *Ortografía de la lengua española*. Espasa.

Figaro (2016a). Operating principle in mos type sensors. `http://www.figarosensor.com/technicalinfo/principle/mos-type.html`.

Figaro (2016b). TGS2600 Product information. `http://www.figarosensor.com/products/2600pdf.pdf`.

Gutiérrez, C. et al. (2016). Desarrollo de una plataforma para discriminación de odorantes mediante técnicas de modulación dinámica. B.S. thesis.

Gutierrez-Osuna, R., Gutierrez-Galvez, A., and Powar, N. (2003). Transient response analysis for temperature-modulated chemoresistors. *Sensors and Actuators B: Chemical*, 93(1):57–66.

Gutierrez-Osuna, R. and Hierlemann, A. (2010). Adaptive microsensor systems. *Annual Review of Analytical Chemistry*, 3:255–276.

Herrero-Carrón, F., Yáñez, D. J., de Borja Rodríguez, F., and Varona, P. (2015). An active, inverse temperature modulation strategy for single sensor odorant classification. *Sensors and Actuators B: Chemical*, 206:555–563.

Hossein-Babaei, F. and Amini, A. (2014). Recognition of complex odors with a single generic tin oxide gas sensor. *Sensors and Actuators B: Chemical*, 194:156–163.

Instrument, T. (2013). BBB Main Page. `https://beagleboard.org/black`.

Instruments, T. (2016). NE555 Timer Texas Instrument. `http://www.ti.com/lit/ds/symlink/ne555.pdf`.

Lee, A. P. and Reedy, B. J. (1999). Temperature modulation in semiconductor gas sensing. *Sensors and Actuators B: Chemical*, 60(1):35–42.

Macías, M. M., Agudo, J. E., Manso, A. G., Orellana, C. J. G., Velasco, H. M. G., and Caballero, R. G. (2013). A compact and low cost electronic nose for aroma detection. *Sensors*, 13(5):5528–5541.

Macías, M. M., Manso, A. G., Orellana, C. J. G., Velasco, H. M. G., Caballero, R. G., and Chamizo, J. C. P. (2012). Acetic acid detection threshold in synthetic wine samples of a portable electronic nose. *Sensors*, 13(1):208–220.

Martinelli, E., Catini, A., and Di Natale, C. (2013). An active temperature modulation of gas sensor based on a self-adaptive strategy. In *Solid-State Sensors, Actuators and Microsystems (TRANSDUCERS & EUROSENSORS XXVII), 2013 Transducers & Eurosensors XXVII: The 17th International Conference on*, pages 2045–2048. IEEE.

Martinelli, E., Polese, D., Catini, A., DAmico, A., and Di Natale, C. (2012). Self-adapted temperature modulation in metal-oxide semiconductor gas sensors. *Sensors and Actuators B: Chemical*, 161(1):534–541.

Pearce, T. C., Schiffman, S. S., Nagle, H. T., and Gardner, J. W. (2006). *Handbook of machine olfaction: electronic nose technology*. John Wiley & Sons.

TGS, F. G. S. (1978). *Tgs2600 figaro engineering inc. Japan (Nov. 1, 1978)*.

Vergara, A., Llobet, E., Brezmes, J., Ivanov, P., Cané, C., Gracia, I., Vilanova, X., and Correig, X. (2007). Quantitative gas mixture analysis using temperature-modulated micro-hotplate gas sensors: Selection and validation of the optimal modulating frequencies. *Sensors and Actuators B: Chemical*, 123(2):1002–1016.

Vergara, A., Llobet, E., Brezmes, J., Vilanova, X., Ivanov, P., Gràcia, I., Cané, C., and Correig, X. (2005). Optimized temperature modulation of micro-hotplate gas sensors through pseudorandom binary sequences. *IEEE Sensors Journal*, 5(6):1369–1378.

Villarreal, D. J. Y. (2009). Estrategias bioinspiradas para la adquisición de olores en narices artificiales. *Master's thesis, Escuela Politécnica Superior, Universidad Autónoma de Madrid*.

Yáñez, D. J., Toledano, A., Serrano, E., de Rosales, A. M. M., Rodríguez, F. B., and Varona, P. (2012). Characterization of a clinical olfactory test with an artificial nose. *Frontiers in neuroengineering*, 5.

A

Instalación del sistema operativo de la BBB

A.1. Instalación del sistema operativo de la BBB

Lo primero que debe hacerse antes de proceder a instalar un sistema operativo nuevo, es hacer un back-up de los datos que tenemos en la BBB, si es que tenemos algún dato, para no perderlos.

A.1.1. Elección del sistema operativo

La BBB tiene varios sistema operativos oficiales que pueden instalarse, estos son algunos:

- Angstrom
- Debian
- ArchLinux
- Gentoo
- Fedora

Estos sistemas se encuentran en la página de BeagleBone: http://beagleboard.org/latest-images. También se puede acceder yendo a la página principal: http://beagleboard.org/ y luego accediendo seleccionando la opción de: Lastest Software Images de la pestaña Start.

Después se mostrará las distintas versiones de los sistemas operativos que se encuentren disponibles en la página web. Para la plataforma se ha descargado la imagen de debian 7.8, para Beagle Bone Black. Es importante comprobar que la versión que descargamos sea compatible nuestra Beagle Bone. Una vez descargada la imagen, hay que descomprimirla ya que viene en un formato .img.xz y se obtendrá la imagen en formato .img y sólo faltaría en grabarla en la tarjeta SD. El programa recomendado por la página oficial de beagle bone para descomprimir la imagen es 7zip (http://www.7-zip.org/download.html).

Para grabar la imagen se necesita un programa especial, ya que la imagen viene en formato binario. Para ello se utiliza el programa Image Write, que se puede descargar en: https://sourceforge.net/projects

/win32diskimager/files/latest/download. Si se usa una distribución linux, existe una serie de comandos que pueden usarse.

Una vez instalado el programa, conectamos la tarjeta microSD al ordenador y abrimos el programa. El programa es muy simple de usar, seleccionamos la imagen y el dispositivo donde guardarla y le damos a: Write. Si sale un mensaje de error indicando que se puede dañar la tarjeta, continuamos.

Por último, antes de proceder a la instalación, insertamos la tarjeta en el ordenador y vamos a la ruta: /media/rootfs/boot/ y abrir el fichero uEnv.txt. Una vez abierto el fichero, buscamos las lineas:

- ##enable BBB: eMMC Flasher:
- #cmdline=init=/opt/scripts/tools/eMMC/init-eMMC-flasher-v3.sh

Borramos la almohadilla(#) de la segunda linea, la que empieza por #cmdline=... guardamos y salimos. Hay que descomentar esa linea para que el SO se instale en la BBB.

Una vez realizados estos pasos, procedemos a instalar el sistema en la BBB. Para ello desconectamos todas las fuentes de corriente de la BBB. Conectamos la tarjeta microSD en la BBB, pulsamos el botón USER/BOOT (figura A.2) y aplicamos corriente por USB o por cable de 5V. Una vez realizados estos pasos, veremos como las luces USRX, figura A.1, de la BBB empiezan a parpadear en orden. Durante este proceso el contenido de la tarjeta SSD se vuelca a la memoria eMMC. Este proceso tiene una duración aproximada de 45 minutos. Cuando se acabe el proceso, estas luces dejarán de parpadear y el proceso se habrá acabado. Desconectamos la micro-SD y la BBB está lista para su uso.

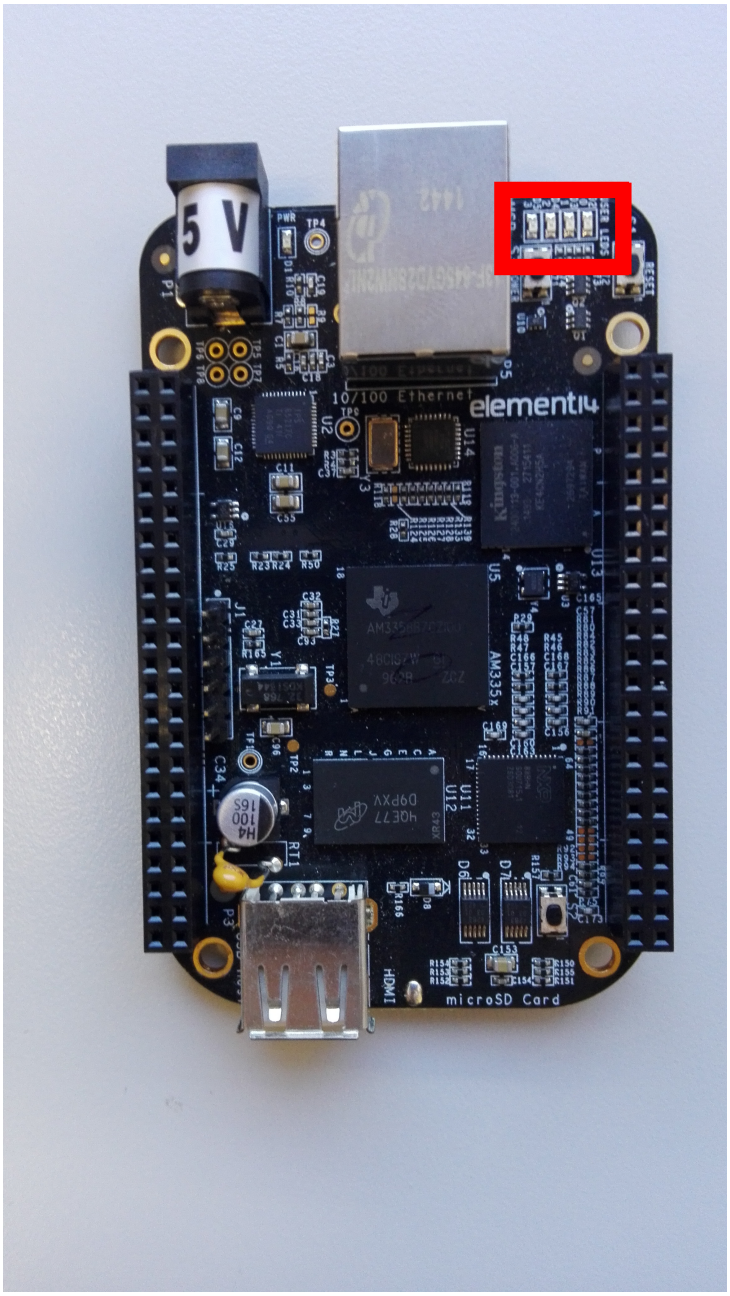


Figura A.1: Imagen de los leds USRX

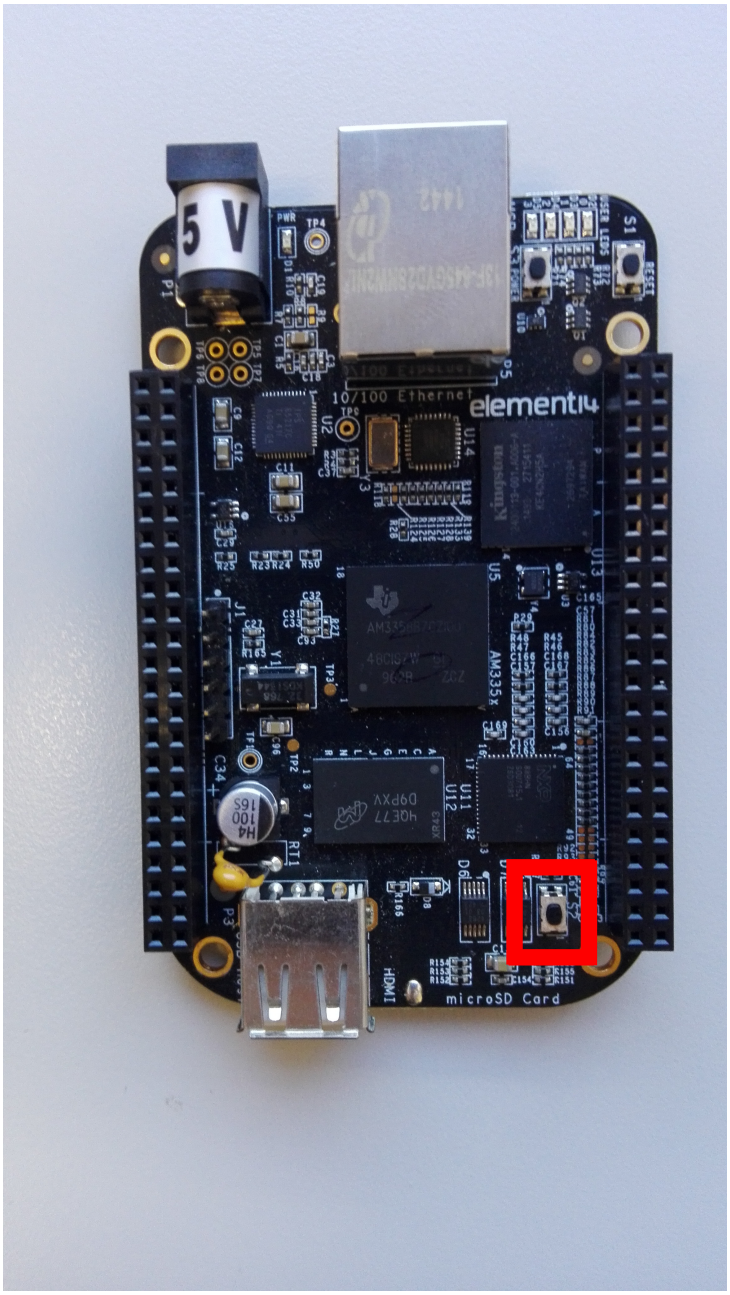


Figura A.2: Imagen del botón USER/BOOT

B

Configuración BBB

B.1. Configuración BBB

Si es necesario la reinstalación del SO en la BBB, los pasos de configuración son los siguientes:

B.1.1. Paso 1 - Conectarse a la BBB

Conectarse por cable USB a la BBB. En una terminal se introduce lo siguiente:
ssh root@192.168.7.2. Como se verá, no pide contraseña. Al conectarse por ssh, se ha conectado como root, que viene sin contraseña. También hay otro usuario, llamado debian. Viene por defecto.

B.2. Paso 2 - Gestión de usuarios

Lo siguiente es crear un usuario nuevo, o usar el usuario debian que viene por defecto, y poner contraseñas a los nuevos usuarios

B.2.1. Cambio de contraseña del super usuario

Una vez logueado como super usuario, hay que poner una contraseña a este usuario. Para ello hay que introducir el siguiente comando:

- passwd

Introducimos la contraseña, dos veces, y con eso ya está cambiada. La contraseña tiene que ser segura, que tenga mayúsculas, minúsculas, caracteres numéricos y símbolos; que no sea demasiado corta pero tampoco muy larga y que se pueda olvidar fácilmente.

B.2.2. Creación de un usuario nuevo

Si se decide a crear un usuario nuevo, hay que introducir los comandos:

- adduser *new_user*

Lo siguiente que nos pedirá será una contraseña para nuestro usuario nuevo. Para ello se recomienda poner una contraseña segura, como se ha dicho anteriormente.

Nos pedirá que repitamos la contraseña, y después pedirá que introduzcamos información adicional, como nombre, apellidos, teléfono, etc. Con introducir como nombre el nombre del usuario nuevo vale. El resto puede dejarse vacío.

Con esto ya tendríamos nuestro usuario nuevo.

Borrar el usuario debian

Si nos hemos creado un usuario nuevo, y no queremos que el usuario debian este creado, podemos borrarlo usando el comando:

- userdel -r debian

Con esto borramos el usuario debian, y su carpeta personal del sistema(bandera -r).

Puede que al intentar borrar el usuario debian, el sistema nos comunique que hay un proceso usando ese usuario. Para poder borrarlo hay que matar ese proceso(kill -9 *id proceso*) y luego volvemos a ejecutar el comando de borrado.

B.2.3. Cambio de contraseña del usuario debian

Si queremos usar el usuario debian, y no crear uno nuevo, es muy importante cambiar la contraseña y no dejar la que viene por defecto. Para ello hay que introducir el comando:

- passwd

Nos pedirá una contraseña, en este caso, como en el anterior, hay que poner una contraseña segura.

Importante Si nos hemos logueado en la BBB con el comando anterior(ssh root@192.168.7.2), estamos en el usuario root. Para cambiar de usuario, se puede hacer de dos formas:

- Saliendo de esa sesión, usando el comando: exit, y entrando con el comando: ssh debian@192.168.7.2. En este caso, este usuario si que pide una contraseña, que es: tmppwd. La introducimos y ya estamos en el usuario debian.

- Ponemos el comando: su debian. Este comando lo que hace es cambiar entre usuarios. Le indicamos que se cambie al usuario debian. Nos pedirá la contraseña; la introducimos y ya estamos en el usuario debian.

B.2.4. Añadir al usuario al grupo de sudo

Para que nuestro usuario, ya sea uno nuevo o el usuario debian por defecto pueda realizar acciones de super usuario, hay que añadirle al grupo sudo. Para ello se pueden usar dos comandos distintos:

- adduser *user* sudo
- usermod -a -G sudo *user*

Importante Para añadir un usuario al grupo sudo, seguramente pida permisos de super usuario. Para ello tenemos que estar logueados como sudo. Puede que haya que reiniciar la BBB para que el cambio surga efecto.

B.3. Configurar interfaz de red y DNS

Para que la BBB tenga acceso a la red y poner conectarse en remoto hay que configurar la interfaz de red y los DNS.

B.3.1. Configuración de la interfaz de red

Para configurar la interfaz de red, hay que cambiar el fichero /etc/network/interfaces, que es el encargado de la interfaz de red. Para cambiar este fichero hay que hacerlo como super usuario.

Para editar el fichero ponemos uno de los siguiente comandos:

- sudo vi /etc/network/interfaces
- sudo nano /etc/network/interfaces

Se abrirá el fichero y podremos observar su contenido, vamos al final del fichero y añadimos las siguientes lineas:

- auto eth0
- iface eth0 inet static
- address *IP asignada*
- netmask *IP de la máscara*
- gateway *IP del gateway*

Guardamos y salimos.

Para los servidores DNS hay dos formas de hacerlo dependiendo de la versión de sistema operativo que tengamos en la BBB. Primero vamos al fichero: /etc/resolv.conf; si en ese fichero hay un mensaje que nos dice que no modifiquemos el contenido porque se borrará, no cambiamos nada; si no nos aparece ningún mensaje, modificamos el fichero con lo siguiente:

- sudo vi /etc/resolv.conf
- sudo nano /etc/resolv.conf

Una vez dentro borramos todo lo que hay en el fichero, y ponemos lo siguiente:

- nameserver: *ip del DNS1 ip del DNS2 ip del DNS3*, ...

Ponemos todas las IPs seguidas por comas.

Si en el fichero nos ponía que no lo modificásemos, las DNS se configuran en el fichero /etc/network/interfaces. Lo abrimos y al final añadimos lo siguiente:

- dns-nameservers *ip del DNS1*,*ip del DNS2*, ...

B.3.2. Cambiar el nombre al host

Para cambiar el nombre al host por el que nos han proporcionado hay que cambiar dos ficheros:

- /etc/hosts
- /etc/hostname

Para el primero, lo abrimos con uno de los dos comandos:

- sudo vi /etc/hosts
- sudo nano /etc/hosts

De este fichero solo interesan las dos primeras lineas:

- 127.0.0.1 localhost
- 127.0.1.1 **beaglebone**.localdomain **beaglebone**

Estas dos primeras lineas empiezan con una IP, ambas. Luego viene información sobre como el sistema tiene configurado los distintos hosts. En este caso, solo nos interesan las palabras en negrita. Hay que sustituirlas por el nombre del host que hayan asignado a la BBB.

Del segundo fichero(/etc/hostname), hay que borrar su contenido(contiene el nombre del host), y escribir el nombre que le hayan aasignado a la BBB.

Con esto ya se habría configurado la interfaz de red y los DNS. Solo falta reiniciar la BBB para que los cambios se apliquen:

- sudo reboot

o usar el siguiente comando para reiniciar la interfaz de red:

- sudo service networking restart

B.4. Configurar el servidor ssh

Para el servidor ssh, es recomendable cambiar ciertos parámetros que vienen configurados por defecto. Para realizar estos cambios hay que cambiar el fichero:

- /etc/ssh/sshd_config

Accedemos al fichero con vi o nano, y buscamos y cambiamos los siguientes parámetros:

- Port X(donde X es cualquier puerto que queramos poner). El puerto estándar es el 22, es muy recomendable cambiar este puerto a otro.
- PermitRootLogin no. Por defecto está a yes, pero es recomendable ponerlo a no y evitar conectarse al usuario root desde ssh
- MaxAuthTries X. El número máximo de intentos antes de que el sistema desautorice esa conexión.
- AllowUsers X. Donde X son los usuarios del sistema que vamos a permitir conectarse por ssh, separados por espacios

Si además queremos usar un certificado público-privado como clave para conectarse, en vez de tener que introducir la contraseña, hay que cambiar o añadir algún campo más:

- RSAAuthentication yes
- PubkeyAuthetication yes
- PasswordAuthentication no

Con estos cambios, ahora solo podría accederse usando un certificado público-privado. **Importante** Si configuramos la BBB para que solo se pueda conectar por certificado es muy importante poner la clave pública del certificado en la BBB en la siguiente ruta, antes de reiniciar el servidor ssh, o la BBB:

- /home/*username*/.ssh/authorized_keys

Si la carpeta no existe, creamos una carpeta nueva con el comando:

- mkdir .ssh

Al empezar por . la carpeta será una carpeta oculta. Dentro de la carpeta, copiamos la clave pública. Si el fichero no existe, lo creamos con el comando:

- touch authorized_keys

Para introducir el contenido de la clave pública en el fichero, si la tenemos en la BBB, usamos el comando:

- cat *clave_publica* » authorized_keys

La explicación de como crear los certificados está en el anexo C.

B.5. Ajustar la hora

La BBB, por defecto trae la hora de Londres. Para cambiar al huso horario deseado, primero tenemos que ir al siguiente directorio:

- /usr/share/zoneinfo/Europe/

Usamos el comando ls, y vemos todos las ciudades que hay en ese directorio. Escogemos la ciudad de la que queramos escoger la hora. En nuestro caso escogemos Madrid. Una vez escogida la ciudad deseada, hay que borrar el fichero:

- /etc/localtime

Seguramente requiera permisos de super usuario. Este fichero es el que regula la hora del sistema. Una vez borrado, hay que crear un enlace simbólico a la ciudad que hayamos elegido. Un enlace simbólico no es más que un fichero que hace referencia a otro fichero, situado en otro lugar del sistema. Para crear el enlace ponemos:

- ln -s /usr/share/zoneinfo/Europe/Madrid /etc/localtime

Tras esto simplemente reiniciamos la BBB:

- sudo reboot

Con esto ya estaría la BBB lista para funcionar.

B.6. Instalación de paquetes Python

Para instalar las bibliotecas de Python necesarias para que los códigos funcionen hay que introducir los siguientes comandos; que son los mismos que se encuentran en (Gutiérrez et al., 2016).

Antes de introducir cualquier comando de ejecución, introducimos el siguiente comando para actualizar los paquetes disponibles:

- sudo apt-get update

Una vez actualizados, procedemos con la instalación de las librerías necesarias

B.6.1. Adafruit_BBIO

Instalamos la librería Adafruit_BBIO, que se trata de una librería libre desarrollada por Adafruit para el control de los pines de la BBB mediante python. Descargamos las dependencias necesarias:

- sudo apt-get install build-essential
  python-dev python-setuptools python-pip python-smbus -y

Para instalar las librerías:

- sudo pip install Adafruit_BBIO

B.6.2. Adafruit_DHT

Esta librería también es de código libre, se encarga de comunicarse con el sensor de temperatura y humedad, y obtiene los valores correspondientes.

Descargamos el código desde Github, y nos movemos a la carpeta donde se encuentra:

- git clone https://github.com/adafruit/Adafruit_Python_DHT.git
- cd Adafruit_Python_DHT

Una vez en la carpeta, instalamos la librería:

- sudo python setup.py install

B.6.3. SciPy

Librería open-source de herramientas y algortimos matemáticos para python. Para instalarla ponemos:

- sudo apt-get install python-scipy

B.6.4. NumPy

Librería libre de python, que contiene funciones para vectores y matrices a alto nivel. Para su instalación ponemos:

- sudo apt-get install python-numpy

C

Creacion de ssh-keys

C.1. Creación de llaves ssh

Para securizar el acceso por ssh a la BBB, se pueden usar certificados público-privados, en vez de contraseñas al conectarse a la BBB. El mecanismo funciona de la siguiente forma: se crean dos ficheros, uno es la clave privada y otro es la clave pública. Para crear estos ficheros, abrimos una terminal e introducimos el siguiente comando:

- ssh-keygen -t *algoritmo* -b *longitud clave*

La opción -t selecciona el algoritmo que va a utilizarse para generar la clave; los distintos algoritmo que proporciona ssh-keygen son:

- rsa1 - La versión 1 del algoritmo rsa
- rsa - La versión 2 del algoritmo rsa
- ecdsa
- ed25519
- dsa

La opción -b indica la longitud, en bits, de la clave. La longitud mínima de la clave es de 1024 bits, y si no se pone nada la clave será de 2048 bits.

Para más información sobre las banderas del comando ssh-keygen, usar el comando: *man ssh-keygen* en una terminal.

Una vez realizado el ssh-keygen, nos mostrará el fingerprint de la clave. Con esto ya tenemos el par de claves. Estas se guardan en el directorio:

- /.ssh/

El nombre de las claves es el siguiente:

- Para la clave privada: id_*nombre del algoritmo*
- Para la clave públic: id_*nombre del algoritmo*.pub

Lo siguiente es copiar la clave pública en la BBB. Para ello usamos el comando scp:

- scp .ssh/id_*nombre del algoritmo*.pub *username*@192.168.7.2:/home/*username*/.ssh/authorized_keys

Con esto ya tenemos la clave pública copiada en el directorio de nuestro usuario.

El funcionamiento del sistema de clave pública-privada es el siguiente: la BBB tiene el certificado público, por lo que cualquier persona puede tratar de establecer una conexión con ella, pero si no tiene la otra parte del certificado, no va a poder conectarse, ya que la BBB le denegará el acceso. Sólo pueden establecer una conexión aquellas personas que posean la otra mitad del certificado, y que como sólo la posee una persona, sabemos a ciencia cierta quien es el que se está conectando. Esta otra mitad hay que dejarla en el sistema con el que queremos a conectarnos a la BBB.

Con esto ya habríamos creado los certificados para la BBB.

D

Cuidado de la BBB

D.1. Cuidado y manejo de la BBB

D.1.1. Manipulación de la BBB

Para realizar cualquier acción con la BBB, antes de tocarla, hay que descargarse de electricidad estática, ya que podríamos quemarla. Para ello con tocar cualquier objeto metálico basta, o con tener una pulsera anti-electricidad estática bastaría.

Si la BBB tiene un circuito montado y se quiere cambiar cualquier elemento de este, es muy importante apagar la BBB, y desconectar los cables de corriente de la BBB. Si no se hace se puede quemar la BBB por un exceso de corriente.

Al montar cualquier circuito, es recomendable asegurarse dos veces que todos los pines están en la posición correcta, ya que es fácil confundirse y estropear la BBB.

Cuando se haya terminado de cambiar todo, volvemos a reconectar los cables de alimentación y podemos volver a usar la BBB.

D.1.2. Pines de la BBB

La BBB tiene, en total, 92 pines de expansión, distribuidos en dos cabezales. Estos pines pueden ser de distintos tipos. Los distintos tipos usados en este proyecto son:

- PWM o Pulse With Modulation. Estos pines son una salida.
- ADC o Analogic to Digital Converter. Estos pines son una entrada.
- GPIO o General Purpose Input/Output. Estos pueden ser tanto una salida como una entrada.

Estos pines tienen un conjunto de limitaciones, que hay que respetar, ya que sino la BBB puede quemarse. Las limitaciones de estos pines se pueden ver en la tabla D.1.

I/O	Nombre	Num Pines	Rango Voltaje	Corriente max	Tiempo Muestreo
Analog (I)	AIN_X	7	0 - 1,8 V	$2\mu A$	$125ns$
Digital (I)	GPIO_X	66	0 o 3.3 V	4 - 6(mA)	140(ns)
Digital (O)	GPIO_X	66	0 o 3,3 V	4 - 6(mA)	95 - 105(ns)

Cuadro D.1: Nomenclatura de los pines de la BBB y sus limitaciones físicas

Si se superan estos rangos, la BBB se estropeará, y habría que adquirir una nueva. Si se monta un circuito en la BBB, será necesario poner algún circuito de protección, si se superan estos límites, ya que se estropearía la BBB.

E

Ejecución programas

E.1. Ejecución de los scripts Python

Para ejecutar un script Python en un sistema Linux, tenemos que tener instalado el intérprete de Python en el sistema. Si no lo tenemos instalado, ejecutamos el siguiente comando:

- sudo apt-get install python

Una vez instalado el intérprete de Python, para ejecutar un script ponemos:

- (sudo) python *nombre_del_script arg1 arg2 ... argN*

El elemento sudo se debe poner cuando sea necesario dar permiso de super usuario al script, porque tiene que acceder a elementos restringidos.

La libreria Adafruit, que es la que se usa para el control de los distintos pines, necesita permisos de super usuario para poder ejecutarse, así que es necesario poner sudo en todas las ejecuciones que se realicen.

E.1.1. Enviar una ejecución a segundo plano

Al ejecutar uno de los script de Python, este tomará el control de la terminal, lo cual puede ser molesto ya que hasta que la ejecución no acabe, no vamos a poder usar esa terminal, y si la cerramos, la ejecución se cortará.

Para enviar una ejecución a segundo plano, y poder seguir usando esa terminal, tenemos que poner al final de la ejecución el símbolo &. Una vez enviado a segundo plano, la terminal nos mostrará un número. Este número es el PID del proceso, y lo usaremos para desvincularlo de la terminal donde lo hemos ejecutado.

Reenviar la salida de pantalla

Si queremos enviar un proceso a segundo plano es necesario que este no utilice la salida estándar: la pantalla. Para que un programa no muestre nada por pantalla es necesario poner al final de la linea de ejecución lo siguiente:

- sudo python PyHuele *ruta_script_ejecucion* > *nombre del fichero* &

El elemento >, lo que hace es redirigir la salida de pantalla al fichero que le indiquemos. Si nuestro programa no utiliza la salida estándar, no es necesario redirigir la salida.

E.1.2. Desvincular el programa de la terminal

Una vez que hemos enviado el programa a segundo plano, falta por desvincularlo de la terminal donde se ha ejecutado. Si ahora se cierra la terminal, el programa finalizaría, ya que este sigue vinculado a esta. Para desvincularlo de la terminal, hay que introducir el siguiente comando:

- disown -h *PID*

Con esto ya hemos desvinculado el proceso de la terminal, y podríamos cerrarla sin temor a que el programa se cortase. Un ejemplo de ejecución sería el siguiente:

- sudo python regresion.py *arg1 arg2 ... argN* > *documento salida* &
- disown -h *PID*

Importante Si al tratar de desvincular el proceso de la terminal, realizarlo como super-usuario.

F

Creacion muestras

F.1. Muestras utilizadas en los experimentos

En los experimentos realizados con la plataforma se han utilizado un conjunto diferente de muestras. Cada conjunto tiene una disolución diferente de los odorantes. Las diferentes muestras utilizadas en los experimentos, ordenadas de mayor concentración a menor concentración, son las siguientes:

- Muestra #0
- Muestra #1
- Muestra #2
- Muestra #3

La muestras se componen de tres odorantes, que son:

- Metanol
- Etanol
- Butanol

También se utiliza aire en las muestras, pero sin diluir su concentración.

Los tres primeros odorantes son los que se diluyen para obtener concentraciones diferentes. Los odorantes se diluyen para ver el efecto de estos en el sensor y evitar que saturase el sensor. Además del odorante, en estado líquido, se ha usado agua destilada para rebajar la concentración del odorante, cada muestra, como se ha dicho con anterioridad, tiene una concentración distinta y es la mitad de la disolución de la muestra anterior; esto es que la muestra #1 tiene la mitad de concentración de los gases de la muestra #0. Antes de utilizar este conjunto de concentraciones de gases, se utilizó otro diferente, que era el que el que se utilizaba en (Gutiérrez et al., 2016). A esta concentración se le llamará #4. Todas las disoluciones de las muestras tienen el mismo volumen y son de 10ml. En la tabla F.1 se muestran las distintas concentraciones del odorante:

Muestra	Metanol	Etanol	Butanol
Muestra #0	10 % (1ml)	100 % (10ml)	2 % (200μl)
Muestra #1	5 % (500μl)	50 % (5ml)	1 % (100μl)
Muestra #2	2'5 % (250μl)	25 % (2'5ml)	0'5 % (50μl)
Muestra #3	1'25 % (125μl)	12'5 % (1'25ml)	0'25 % (25μl)
Muestra #4	50 % (5ml)	5 % (5ml)	2 % (200μl)

Cuadro F.1: Distintas concentraciones en % del odorante usadas como muestras. Todos los botes tienen 10ml.

La última muestra, #4, no sigue el mismo patrón de concentración de odorantes que se usa con el resto de las muestras.

Para realizar las muestras se ha utilizado una micro-pipeta regulable que permite escoger la cantidad de líquido que va a introducirse en la muestra y dos pipetas F.1.

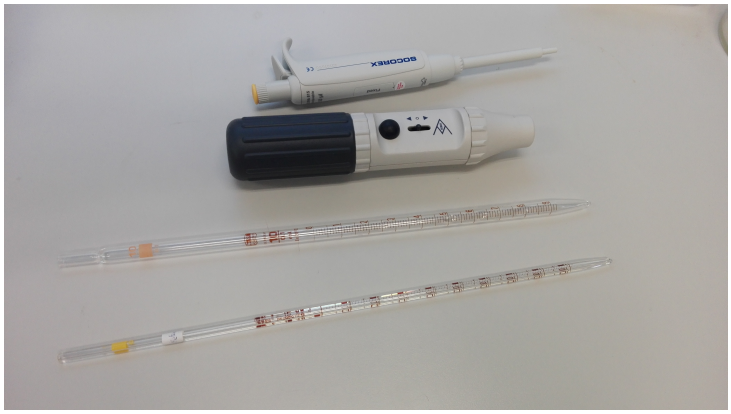


Figura F.1: Imagen las pipetas

El objeto superior es la micro-pipeta y los dos tubos de vidrio son las pipetas. El objeto que está debajo de la micro-pipeta es una bomba que introduce el líquido en la pipeta.

En la figura F.1 se muestran imágenes de los odorantes para ver sus concentraciones.

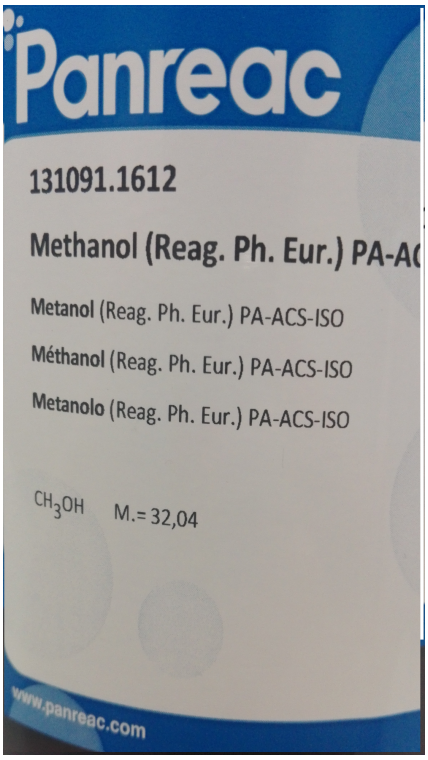


Figura F.2: Metanol. Concentración del 99'8 %

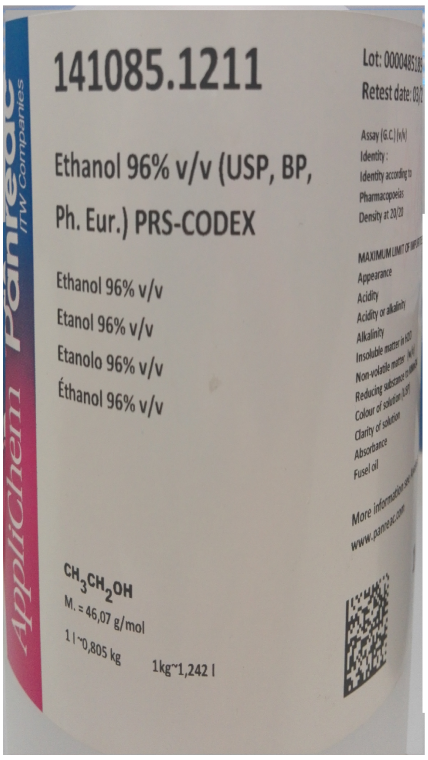


Figura F.3: Etanol. Concentración del 96 %

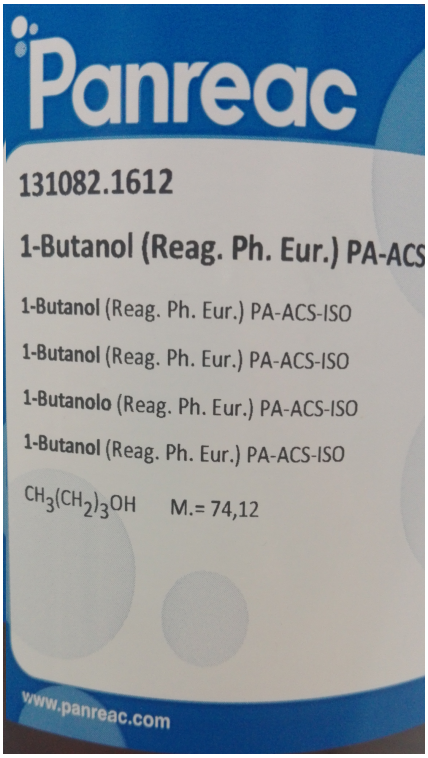


Figura F.4: Butanol. Concentración del 99'8 %

G

Ficheros ejecución

G.1. Ficheros ejecución

A continuación se muestran los diferentes ficheros de ejecución con las configuraciones usadas para capturar las muestras de odorantes en la plataforma.

G.1.1. Script para la captura de datos método de codificación en frecuencia

```
//Script para la ejecución de 5 capturas con Martinelli
//La modulación escogida es Martinelli
modulacion: 3
//Se desean hacer 6 experimentos diferentes
numero_experimentos: 6
//La succión en todos es del 70\%
succion: 70*
//Cada gas se analiza 10 transiciones de estados
duracion_estimulo: 10*
//10 muestras iniciales
samples_inicio: 10*
//3 gases aleatorios para un experimento
numero_muestras: 3
//Sin aire entre gases
segundos_entre_estimulos: 0*
//Nombre de los archivos, el último es común a todas las capturas
nombre_archivo: calentamiento,martinelli_capturas*
//Nombre de las carpetas, el último es común a todas las capturas
nombre_carpeta: Calentamiento,Martinelli*
//Versión de los experimentos: aleatoria y todas las transiciones
version_experimento: 2,1*
//Modo de las capturas por transiciones entre estados
modo_martinelli_ejecucion: 2*
//Modo de variación de la temperatura de forma gradual
```

```
modo_martinelli_temperatura: 2*
//La temperatura mínima
temperatura_minima_martinelli: 60*
//La temperatura máxima
temperatura_maxima_martinelli: 100*
```

Este fichero se ha utilizado en la ejecución de experimentos de captura de datos usando el algoritmo de temperatura variable y codificación en frecuencia. Tiene 6 ejecuciones en total. En todas las ejecuciones, los parámetros siguientes son los mismos:

- Succión al 70 %.
- Duración de 10 transiciones para cada gas.
- 10 muestras iniciales.
- Sin muestras entre estímulos.
- El modo de variación de temperatura es síncrono a los pulsos, aumentando o disminuyendo un poco la temperatura cada pulso.
- La temperatura mínima y máxima es de 60 y 100.

La primera ejecución es aleatoria y tiene 3 muestras aleatorias. En el resto de capturas se analizan todas las transiciones posibles. La primera captura se realiza siempre para calentar el sensor, y no se representa, ya que al estar el sensor frío salen valores distintos.

El script usado para las capturas que tiene un tiempo fijo es el siguiente:

```
//Script para la ejecución de 10 capturas con Martinelli
//La modulación escogida es Martinelli
modulacion: 3
//Se desean hacer 10 experimentos diferentes
numero_experimentos: 10
//La succión en todos es del 70\%
succion: 70*
//Cada gas se analiza 120 segundos
duracion_estimulo: 120*
//10 muestras iniciales
samples_inicio: 10*
//4 gases aleatorios para un experimento
numero_muestras: 4
//Sin aire entre gases
segundos_entre_estimulos: 0*
//Nombre de los archivos, el último es común a todas las capturas
nombre_archivo: calentamiento,martinelli_capturas*
//Nombre de las carpetas, el último es común a todas las capturas
nombre_carpeta: Calentamiento,Martinelli*
//Versión de los experimentos: aleatoria y todas las transiciones
version_experimento: 2,1*
//Modo de las capturas por tiempo
modo_martinelli_ejecucion: 1*
//Modo de variación de la temperatura de forma brusca
modo_martinelli_temperatura: 1*
```

```
//La temperatura mínima
temperatura_minima_martinelli: 60*
//La temperatura máxima
temperatura_maxima_martinelli: 100*
```

Este script tiene las siguientes características:

- Succión al 70 %.
- Duración de 120 segundos para cada gas.
- 10 muestras iniciales.
- Sin muestras entre estímulos.
- El modo de variación de temperatura se produce de forma brusca entre el mínimo y el máximo.
- La temperatura mínima y máxima es de 60 y 100.

G.1.2. Script para la captura de datos método de temperatura fija

El primero de los scripts para la captura de los datos con el algoritmo de temperatura fija:

```
//Script para la ejecución de 10 capturas con puro
//La modulación escogida es puro
modulacion: 1
//Se desean hacer 11 experimentos diferentes
numero_experimentos: 11
//La succión en todos es del 70\%
succion: 70(11)
//10 gases aleatorios para un experimento
numero_muestras: 10
//Cada gas se analiza 120 segundos
duracion_estimulo: 60*
//30 muestras iniciales
muestras_iniciales: 30*
//Sin aire entre gases
segundos_entre_estimulos: 0*
//Nombre de los archivos, el último es común a todas las capturas
nombre_archivo: calentamiento,capturas(10)
//Nombre de las carpetas, el último es común a todas las capturas
nombre_carpeta: Calentamiento,Capturas(10)
//Versión de los experimentos: aleatoria y todas las transiciones
version_experimento: 2,1(10)
```

Este fichero se utilizó en la captura de datos con el algoritmo de temperatura fija. Tiene un total de 11 ejecuciones, las características del experimento son:

- Succión al 70 %.
- Duración de 60 segundos para cada gas.

- 30 muestras iniciales.
- Sin muestras entre estímulos.
- Para la primera captura son gases aleatorios, 10 gases y para el resto son todas las transiciones de los odorantes seguidas.

Para las capturas de la representación 5.1, se usó el siguiente script:

```
//Script para la ejecución de 3 capturas con puro
//La modulación escogida es puro
modulacion: 1
//La succión en todos es del 70\%
succion: 70(3)
//Se desean hacer 4 experimentos diferentes
numero_experimentos: 4
//4 gases aleatorios para un experimento
numero_muestras: 4
//Cada gas se analiza 90 segundos
duracion_estimulo: 90*
//90 muestras iniciales
muestras_iniciales: 90*
//Con 90 segundos de aire entre gas y gas
segundos_entre_estimulos: 90*
//Nombre de los archivos, el último es común a todas las capturas
nombre_archivo: calentamiento,puro(3)
//Nombre de las carpetas, el último es común a todas las capturas
nombre_carpeta: Calentamiento,Puro(3)
//Versión de los experimentos: aleatoria e introducida
//por el usuario
version_experimento: 2,3*
//La secuencia de gases elegida para 3 ejecuciones
vector_apertura_valvulas: 2-3-1(3)
```

El script es para 4 capturas, la primera para calentar el sensor y el resto es para captar odorantes. Las características son las siguientes:

- Succión al 70 %.
- Duración de 90 segundos para cada gas.
- 90 muestras iniciales.
- 90 segundos de estabilización entre estímulos.

G.1.3. Script para la captura de datos con el método de temperatura variable y codificación en amplitud

El script de captura de los datos es el siguiente:

```
//Script para la ejecución de 3 capturas con regresión
//La modulación escogida es regresion
modulacion: 2
```

```
//La succión en todos es del 70\%
succion: 70*
//Se desean hacer 3 experimentos diferentes
numero_experimentos: 3
//5 gases aleatorios para un experimento
numero_muestras: 5
//Cada gas se analiza 120 segundos
duracion_estimulo: 120*
//10 muestras iniciales para las 2 primeras ejecuciones
//y 30 para la ultima
muestras_iniciales: 10(2),30
//Con 0 segundos de aire entre gas y gas
segundos_entre_estimulos: 0*
//Nombre de los archivos, el último es común a todas las capturas
nombre_archivo: calentamiento,capturas*
//Nombre de las carpetas, el último es común a todas las capturas
nombre_carpeta: Calentamiento,Capturas*
//Versión de los experimentos: aleatoria y todas las transiciones
version_experimento: 2,1(2)
//La tendencia es de 5 para todas las transiciones
tendencia: 5*
//El calentamiento del sensor es al 50\% del total
calentamiento_sensor: 50*
```

Los datos capturas mediante este modo tienen todos las mismas características, que son las siguientes:

- Succión al 70 %.
- Duración de 120 segundos para cada gas.
- 10 muestras iniciales las dos primeras capturas y 30 la última.
- Sin muestras entre estímulos.
- La misma tendencia para todas las capturas: 5.
- Temperatura al 50 % de 5V.

El script para la captura de la imagen 2.5 es el siguiente:

```
//Script para la ejecución de 2 capturas con regresión
//La modulación escogida es regresion
modulacion: 2
//La succión en todos es del 70\%
succion: 70
//Se desean hacer 2 experimentos diferentes
numero_experimentos: 2
//5 gases aleatorios para un experimento
numero_muestras: 5
//Cada gas se analiza 30 segundos
duracion_estimulo: 30*
//10 muestras iniciales para las 2 primeras ejecuciones
```

```
//y 30 para la ultima
muestras_iniciales: 10*
//Con 60 segundos de aire entre gas y gas
segundos_entre_estimulos: 60*
//Nombre de los archivos, el último es común a todas las capturas
nombre_archivo: calentamiento,capturas\_regresion
//Nombre de las carpetas, el último es común a todas las capturas
nombre_carpeta: Calentamiento,Captura\_Regresion
//Versión de los experimentos: aleatoria y elegida por el usuario
version_experimento: 2,3
//La tendencia es de 5 para todas las transiciones
tendencia: 5*
//El calentamiento del sensor es al 50\% del total
calentamiento_sensor: 50*
//La secuencia de gases elegida
vector_apertura_valvulas: 1-2-3
```

Las características de este tipo de ejecución son:

- Succión al 70 %.
- Duración de 30 segundos para cada gas.
- 10 muestras iniciales.
- 60 muestras entre estímulos.
- La misma tendencia para todas las capturas: 5.
- Temperatura al 50 % de 5V.

El script de captura de los datos es el siguiente:

```
//Script para la ejecución de 10 capturas con regresión
//La modulación escogida es regresion
modulacion: 2
//La succión en todos es del 70\%
succion: 70*
//Se desean hacer 11 experimentos diferentes
numero_experimentos: 11
//5 gases aleatorios para un experimento
numero_muestras: 5
//Cada gas se analiza 60 segundos
duracion_estimulo: 60*
//30 muestras iniciales
muestras_iniciales: 30*
//Con 0 segundos de aire entre gas y gas
segundos_entre_estimulos: 0*
//Nombre de los archivos, el último es común a todas las capturas
nombre_archivo: calentamiento,capturas*
//Nombre de las carpetas, el último es común a todas las capturas
nombre_carpeta: Calentamiento,Capturas*
//Versión de los experimentos: aleatoria y todas las transiciones
version_experimento: 2,1(10)
```

```
//La tendencia es de 5 para todas las transiciones
tendencia: 5*
//El calentamiento del sensor es al 50\% del total
calentamiento_sensor: 50*
```

Los datos capturas mediante este modo tienen todos las mismas características, que son las siguientes:

- Succión al 70 %.
- Duración de 60 segundos para cada gas.
- 30 muestras iniciales en todas las capturas.
- Sin muestras entre estímulos.
- La misma tendencia para todas las capturas: 5.
- Temperatura al 50 % de 5V.

H

Gráficas de las ejecuciones

H.1. Gráficas de las experimentaciones

En la figura H.1 se muestran más capturas tomadas usando el algoritmo de temperatura constante. Como se observa en las figuras, los valores obtenidos varían en 60mV como máximo. Al igual que las capturas mostradas en el capítulo 5, estos resultados son satisfactorios, ya que se observa una gran reproducibilidad. Cada figura tiene 10 capturas diferentes y cada captura está representada por un color distinto. Todas las capturas tiene los mismos parámetros: 60 segundos analizando cada gas y a misma secuencia de gases: muestras iniciales, metanol, metanol, etanol, etanol, butanol, butanol, aire, aire, etanol, metanol, butanol ,metanol, aire, butanol, etanol, aire y metanol. Todas las figuras de este anexo siguen el mismo patrón de transiciones.

Las figuras muestran los valores captados por el sensor, a la izquierda, y a la derecha los valores de la resistencia del sensor. Como puede verse en las figuras, se tienen diferentes valores máximos en las capturas del sensor. Esto puede deberse a varios efectos, como el drift del sensor, el tiempo que el sensor lleva usándose, cuanto más tiempo mejor, la suciedad que se acumula en el sensor o la evaporación de los gases, entre otras causas.

La figura H.2 muestra capturas realizadas con el algoritmo de captura de datos con temperatura variable y codificación en amplitud. Los resultados también son satisfactorios. La concentración usada es #0.

Como puede verse en la figura H.2, los valores tienen un nivel de reproducción más altos que con las capturas de temperatura fija; obteniendo una diferencia de 5 mV como máximo, en el caso del valor captado por el sensor y de 0,25V en el caso de la temperatura del sensor, lo que demuestra que la captura con codificación en amplitud tiene menor variabilidad que con temperatura fija como ya se indicó en el capítulo 5.

En total son 10 capturas diferentes, donde cada captura se representa con un color diferente.

Las figuras de la tabla H.3 muestran las capturas del modelo propuesto por el profesor Martinelli, usando una ventana de tiempo variable. Como se explicó anteriormente, los pulsos pueden no llegarse a completar y no se pueden comparar los pulsos completos entre sí.

Cada figura muestra las transiciones de un odorante al resto, y los títulos de las figuras se indican las transiciones. También se ha añadido la media y la varianza de las transiciones entre

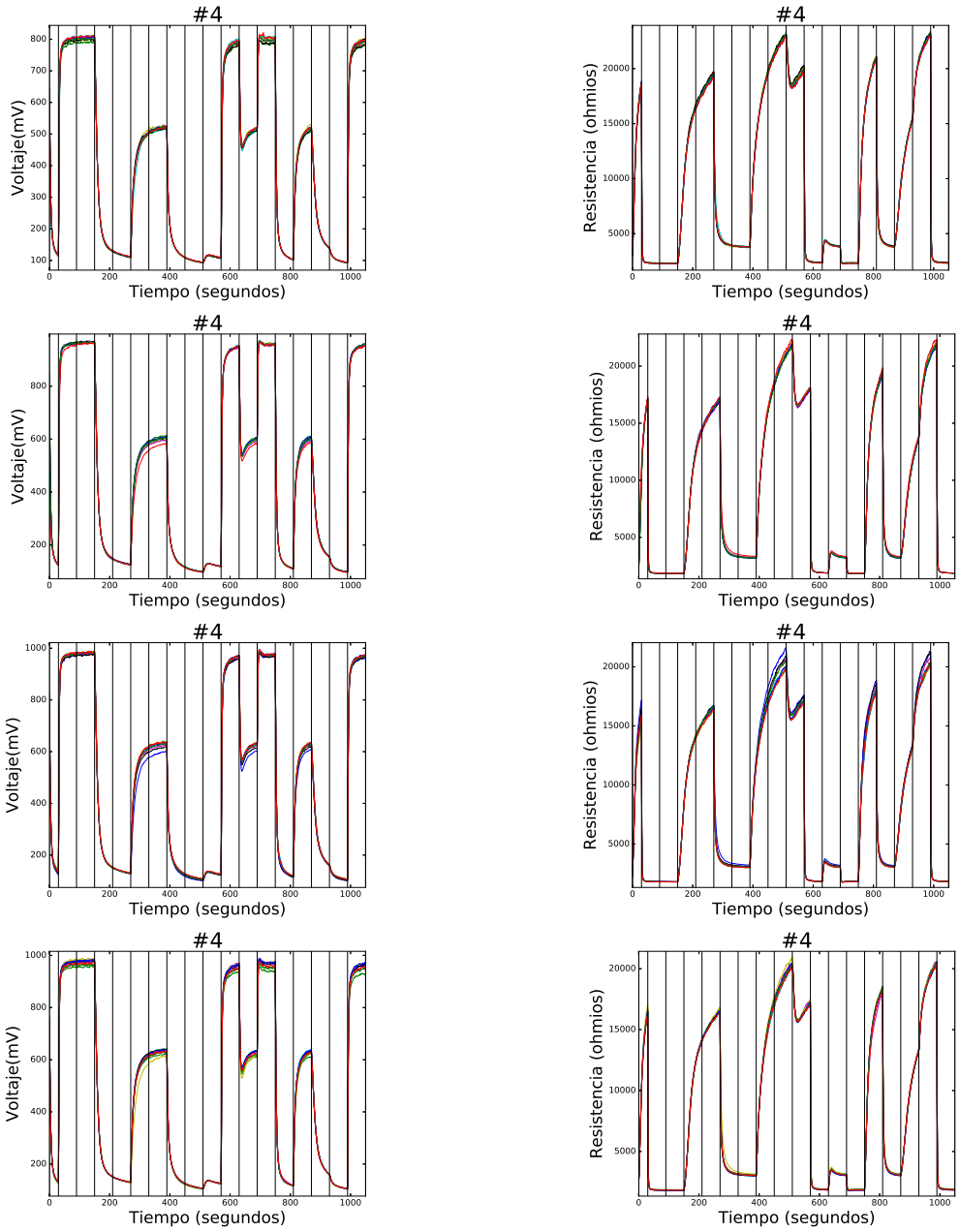


Figura H.1: Capturas de diferentes gases con el algoritmo de temperatura constante. La muestra usada es la #4. Cada gas se analizó durante 60 segundos y cada captura tiene la secuencia: muestras iniciales, metanol, metanol, etanol, etanol, butanol, butanol, aire, aire, etanol, metanol, butanol ,metanol, aire, butanol, etanol, aire y metanol

estados completas y de los estados alto y bajo que se han quedado incompletos.

Como se observa en la figura H.3, las transiciones obtenidas por el método de tiempo constante tienen medias y varianzas muy parecidas. Observando estos resultados se determina, que para 120 segundos, el tiempo es un factor que dificulta la clasificación, y que para ventanas de tiempo mayores el resultado puede mejorar.

Para el algoritmo de temperatura variable y codificación en frecuencia, se tienen los histogramas de las distintas concentraciones utilizadas por separado H.4. Cada histograma contiene 16 paneles, que corresponden a todas las transiciones de los odorantes. Los paneles se han representado entre el mínimo y el máximo de cada transición para ver la variabilidad de los gases.

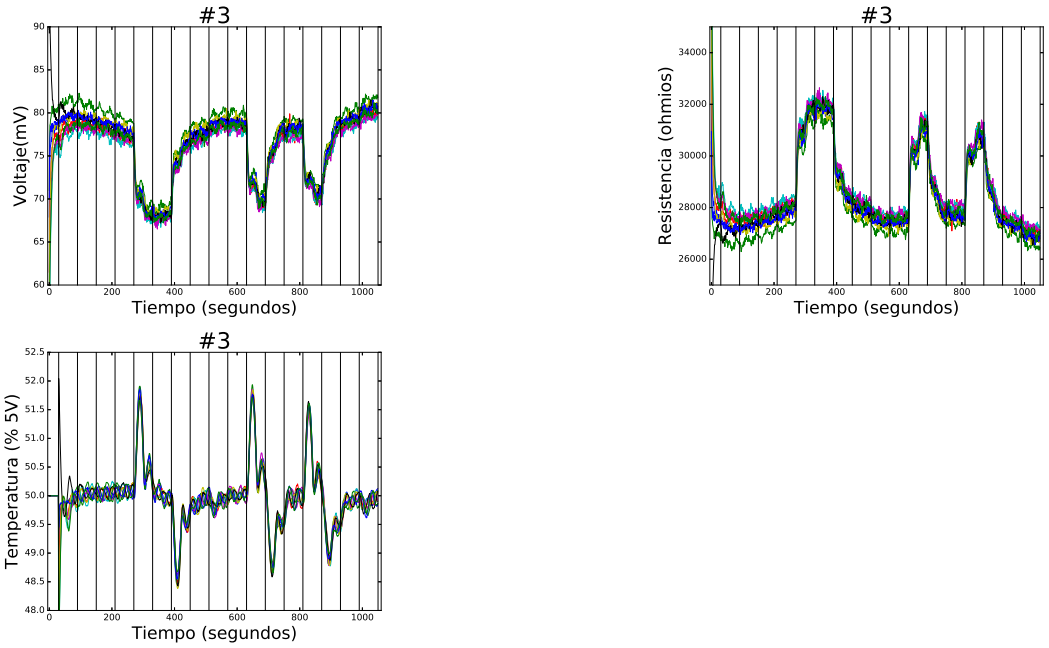


Figura H.2: Capturas de diferentes gases con el algoritmo de temperatura variable y codificación en amplitud. La concentración usada es la #3. Cada gas se analizó durante 60 segundos y cada captura tiene la secuencia: muestras iniciales, metanol, metanol, etanol, etanol, butanol, butanol, aire, aire, etanol, metanol, butanol ,metanol, aire, butanol, etanol, aire y metanol

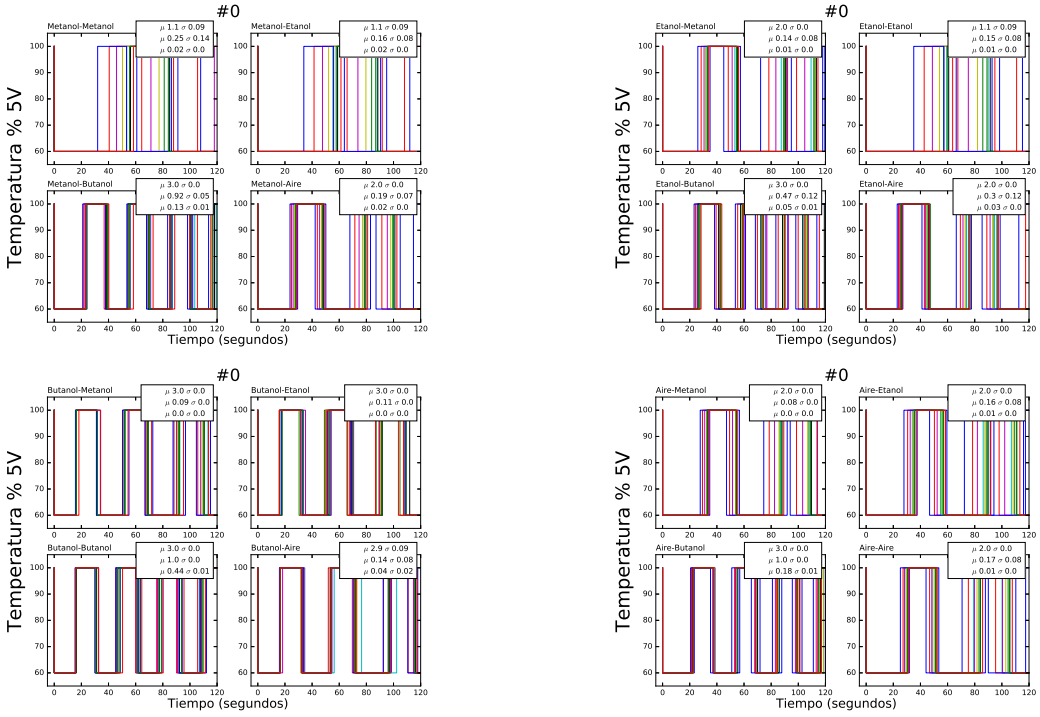


Figura H.3: Capturas de datos con el algoritmo de temperatura variable y codificación en frecuencia.Captura para ver la señal de la temperatura con diferentes muestras iniciales. La secuencia de gases es: muestras iniciales, metanol, metanol, etanol, etanol, butanol, butanol, aire, aire, etanol, metanol, butanol ,metanol, aire, butanol, etanol, aire y metanol. La concentración es #0, con 120 segundos para cada odorante y el script para está en G.1.1.

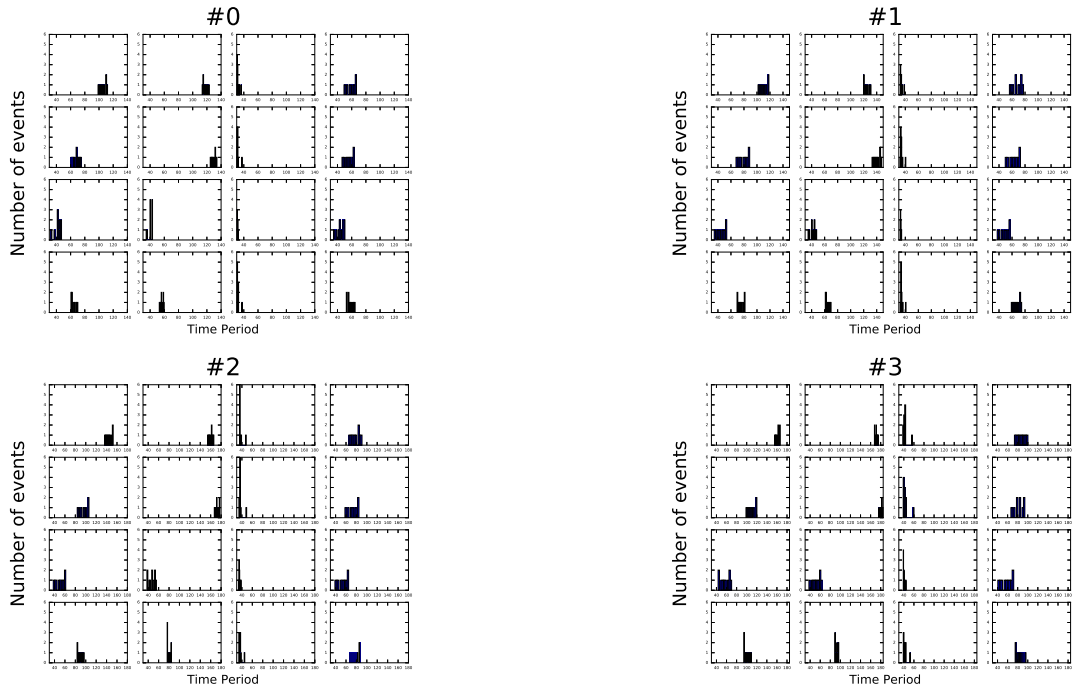


Figura H.4: Histogramas de las transiciones de los diferentes gases. Los paneles están ordenados por las transiciones según las filas y las columnas de la siguiente forma: metanol,etanol,butanol y aire.El script de ejecución es G.1.1

Cada histograma representa el tiempo que se ha necesitado en captar las 10 transiciones entre estados. En cada histograma se representa una concentración diferente. Con cada concentración se ha realizado 5 capturas distintas y se han promediado para representarlas todas. Los paneles de cada histograma están ordenados, de forma que se puedan ver las transiciones en un orden. Las filas tienen el siguiente orden: metanol, etanol, butanol y aire y las columnas el siguiente: metanol, etanol, butanol y aire, formando una matriz.

I

Reunificación de código

I.1. Reunificación de código

En este anexo se explican los cambios realizados en los códigos iniciales y que han sido utilizados en el desarrollo de **PyHuele**. Todo este código pertenece al módulo de captura de datos. A estos ficheros se les ha aplicado varios cambios acortando las lineas de código y creando una librería nueva.

Lo primero que se hizo fue reducir el número de lineas, unificando varias funciones con funcionalidades muy parecidas en una sola. Para ello se ha tenido que aumentar el número de argumentos en las funciones.

Todo el código que era común a los distintos tipos de modulación se ha agrupado en un sólo archivo. En ese archivo se incluyen las funciones de: apertura de válvulas, creación de ficheros, manejo de las electroválvulas y la función de medición de temperatura y humedad .

Los ficheros Python de los que se dispone en la actualidad son:

- Puro.py - Contiene las funciones de ejecución del algoritmo que de temperatura constante.
- Regresion.py - Contiene las funciones de ejecución del algoritmo de temperatura variable y codificación en amplitud.
- Martinelli.py - Contiene las funciones de ejecución del algoritmo de temperatura variable y codificación en frecuencia.
- EN_codigo_comun.py - Contiene las funciones comunes de los distintos algoritmos de modulación. Este fichero es nuevo.

A continuación se procede a explicar las mejoras realizadas sobre el código.

I.2. Mejora de funciones

Entre las funciones que se han mejorado se encuentran las siguientes:

- Apertura de válvulas: Este es el código que había antes para la apertura de electroválvulas:

```
def apertura(electrovalvula):
    if electrovalvula == 1:
        GPIO.output(electrovalve1,GPIO.HIGH)
        GPIO.output(electrovalve2,GPIO.LOW)
        GPIO.output(electrovalve3,GPIO.LOW)
        GPIO.output(electrovalve4,GPIO.LOW)
    elif electrovalvula == 2:
        GPIO.output(electrovalve1,GPIO.LOW)
        GPIO.output(electrovalve2,GPIO.HIGH)
        GPIO.output(electrovalve3,GPIO.LOW)
        GPIO.output(electrovalve4,GPIO.LOW)
    elif electrovalvula == 3:
        GPIO.output(electrovalve1,GPIO.LOW)
        GPIO.output(electrovalve2,GPIO.LOW)
        GPIO.output(electrovalve3,GPIO.HIGH)
        GPIO.output(electrovalve4,GPIO.LOW)
    elif electrovalvula == 4:
        GPIO.output(electrovalve1,GPIO.LOW)
        GPIO.output(electrovalve2,GPIO.LOW)
        GPIO.output(electrovalve3,GPIO.LOW)
        GPIO.output(electrovalve4,GPIO.HIGH)
```

Esta función dividió en dos, una para la apertura, y otra para el cierre. La mejora realizada permitió que se pudiesen abrir varias válvulas simultáneamente, ya que antes soló se podía abrir una válvula a la vez. Para ello, en vez de un número, que representa una válvula, se le pasa una lista, con las válvulas que se desean abrir.

```
def abrir_electrovalvulas(electrovalvulas):
    for electrovalvula in electrovalvulas:
        GPIO.output(apertura_electrovalvulas
            [electrovalvula], GPIO.HIGH)
```

Para el cierre de válvulas, se cierran todas las válvulas. Es lo más simple que se puede hacer, ya que ir comprobando que válvula está abierta y cerrarla es más pesado y costoso, además que sería inviable para el caso de que hubiese muchas válvulas.

```
def cerrar_electrovalvulas():
    GPIO.output(electrovalve1, GPIO.LOW)
    GPIO.output(electrovalve2, GPIO.LOW)
    GPIO.output(electrovalve3, GPIO.LOW)
    GPIO.output(electrovalve4, GPIO.LOW)
```

- Función de creación del fichero de salida. Esta función está repetida en los tres archivos Python que había anteriormente. Se ha unificado para que solo sea una.

  Antes de modificar las funciones, había tres con el siguiente estilo:

```
def file_TGS2600(vec_open_valve,succion,heat2600,SAMPLES):
    f = open(ruta_fichero, "w+")
    f.write ('////////////////////////////
        ////////////////////////////////////////\n')
    f.write ('\n\nPlataforma de experimentacion:
```

```
        \n\nSERGIO DE LA CRUZ GUTIERREZ\n\n')
    f.write ('Sensor TGS2600\n')
    f.write ('Algoritmo MARTINELLI\n')
    date=time.strftime("%a%d%b%Y-%HH%MM%SS", time.localtime())
    f.write ('Fecha y hora de inicio: ' + str(date) + '\n')
    f.write ('Ruta del fichero: ' + str(ruta) + '\n\n')
    f.write ('Nombre del fichero: ' + str(fileString) + '\n')
    f.write ('Nombre del fichero de datos 1: '
        + str(fileString_data1)+ '\n')
    f.write ('Nombre del fichero de datos 2: '
        + str(fileString_data2)+ '\n\n')
    f.write ('Nombre del fichero de TyH: '
        + str(nameFile_tyh) + '\n')
    f.write ('Electrovalvula (1-METANOL, 2-ETANOL,
        3-BUTANOL, 4-AIRE )\n\n')
    f.write ('Conmutacion entre electrovalvulas: '
        +str(vec_open_valve)+ '\n\n')
    f.write ('Parametros para la experimetacion, +
        se introducen como argumentos \n')
    f.write ('Succion motor (50-100)>>> '
        + str(succion)+ '\n')
    f.write ('Duracion del experimento: '
        +str(SAMPLES)+ ' muestras\n')
    f.write
    ('////////////////////////////////////
        ////////////////////////////////////\n\n')
    f.write (' CAPTURA DE DATOS:\n\n')
```

La nueva función es la siguiente:

```
def file_TGS2600(vec_open_valve,succion,heat2600,tendencia,
    heatPin2600,tiempo,switch,algoritmo,ruta,nameFile,
    nameFile_data1,nameFile_data2,nameFile_tyh,f,opcion
    ,tespera,SAMPLESINICIO):
  f.write ('////////////////////////////////////
      ////////////////////////////////////\n')
  f.write ('Plataforma de experimentacion:
      \n\nSERGIO DE LA CRUZ GUTIERREZ\n')
  f.write ('Sensor TGS2600\n')
  f.write ('Algoritmo: ' + algoritmo + '\n')
  date = time.strftime("%a%d%b%Y-%HH%MM%SS",
      time.localtime())
  f.write ('Fecha y hora de inicio: '
      + str(date) + '\n')
  f.write ('Ruta del fichero: '
      + str(ruta) + '\n')
  f.write ('Nombre del fichero: '
      + str(nameFile) + '\n')
  f.write ('Nombre del fichero de datos 1: '
      + str(nameFile_data1) + '\n')
  if opcion == 2:
    f.write ('Nombre del fichero de datos 2: '
```

```
        + str(nameFile_data2) + '\n')
  f.write ('Nombre del fichero de TyH: '
      + str(nameFile_tyh) + '\n')
  f.write ('Electrovalvula (1-METANOL, 2-ETANOL,
      3-BUTANOL, 4-AIRE ) \n')
  f.write ('Conmutacion entre electrovalvulas: '
      +str(vec_open_valve)+ '\n\n')
  f.write ('Parametros para la experimetacion,
      se introducen como argumentos \n')
  f.write ('Succion motor (30-100)
      >>> '+ str(succion)+ '%\n')

  if opcion == 1:
    f.write ('Temperatura Promedio TGS2600 (1-100)
        >>> '+str(heat2600)+'% Pin PWM : '
        +str(heatPin2600)+ '\n')
    f.write ('La tendencia que va a
        utilizarse: '+str(tendencia)+'\n')

  f.write ('Se ha dejado un tiempo entre
      captura de gases de: '+str(tespera)
      + ' segundos\n')
  f.write ('El experimento tiene: '
      +str(SAMPLESINICIO)+ ' muestras iniciales\n')
  f.write ('Duracion del experimento: '
      +str(tiempo)+ ' minutos o muestras\n')
  f.write ('El tiempo de switch o conmutacion
      de las electroválvulas es de: '+str(switch)+'\n')
  f.write ('////////////////////////////
      ////////////////////////////////////////\n')
  f.write ('CAPTURA DE DATOS:\n\n')
```

Como puede verse, la nueva función contiene más información, como el tiempo de captura entre gases, el número de muestras iniciales, etc. La nueva función tiene más argumentos para que pudiese usarse con los diferentes algoritmos de captación de odorantes.

- Creación de directorios y nombres de ficheros. Esta función también se creó para unificar código, repetido en todos los archivos Python.

```
nameFile = "Martinelli_TGS2600.txt"
nameFile_data1 = "Martinelli_data1.dat"
nameFile_data2 = 'Martinelli_data2.dat'
nameFile_tyh = "TyH_TGS2600.dat"
fileString = time.strftime("%a%d%b%Y-%HH%MM%SS",
    time.localtime())+'_'+nameFile
fileString_data1 = time.strftime("%a%d%b%Y-%HH%MM%SS",
    time.localtime())+'_' +nameFile_data1
fileString_data2 = time.strftime("%a%d%b%Y-%HH%MM%SS",
    time.localtime())+'_' +nameFile_data2
fileString_tyh = time.strftime("%a%d%b%Y-%HH%MM%SS",
    time.localtime())+'_' +nameFile_tyh
ahora = datetime.now()
```

```
ruta  ="/media/MICROSD/FRECUENCIA/MARTINELLI/"
    +str(ahora.month)+"/"+str(ahora.day)+"/"
if not os.path.exists(ruta):
    os.makedirs(ruta)

ruta_fichero = ruta.strip() + str(fileString)
ruta_fichero_data1 = ruta.strip() + str(fileString_data1)
ruta_fichero_data2 = ruta.strip() + str(fileString_data2)
ruta_fichero_tyh = ruta.strip() + str(fileString_tyh)
```

Este código se encontraba en todos los ficheros de los algoritmos de captación de odorantes, variando los nombres de los archivos. Para que se pueda usar con cualquier algoritmo, se ha encapsulado en una función.

```
def create_files(nameFile,path,folder,samplesinicio,ahora):
    fileString = nameFile+"_txt_" +
        time.strftime("%a%d%b%Y-%HH%MM%SS",
        time.localtime())+".txt"
  fileString_data = nameFile+"_dat1_"+
      time.strftime("%a%d%b%Y-%HH%MM%SS",
        time.localtime())+".dat"
  fileString_tyh = "TyH"+nameFile+"_data_"+
      time.strftime("%a%d%b%Y-%HH%MM%SS",
      time.localtime())+".data"

  ruta1 = path + "/" + str(ahora.year)+str(ahora.month)+
      str(ahora.day) + "/" + folder + "/" +
      str(samplesinicio) + "SAMPLESINICIO" + "/" + "dat"
      + "/"
  if not os.path.exists(ruta1): os.makedirs(ruta1)
  ruta2 = path + "/" + str(ahora.year)+str(ahora.month)+
      str(ahora.day) + "/" + folder + "/" +
      str(samplesinicio) + "SAMPLESINICIO" + "/" + "txt"
      + "/"
  if not os.path.exists(ruta2): os.makedirs(ruta2)
  ruta3 = path + "/" + str(ahora.year)+str(ahora.month)+
      str(ahora.day) + "/"  + folder + "/" +
      str(samplesinicio) + "SAMPLESINICIO" + "/" + "TyH"
      + "/"
  if not os.path.exists(ruta3): os.makedirs(ruta3)

  ruta_fichero = ruta2.strip() + str(fileString)
  ruta_fichero_data = ruta1.strip()+str(fileString_data)
  ruta_fichero_tyh = ruta3.strip() + str(fileString_tyh)
  return ruta_fichero,ruta_fichero_data,ruta_fichero_tyh
```

Esta función se puede utilizar ahora para cualquier modulación, ya que se ha puesto todas las opciones configurables en los argumentos.

- Aperturas de los ficheros. Esta función se creó por simplicidad y para reducir el código en el resto de ficheros.

```
def open_files(ruta_fichero,ruta_fichero_data,
```

```
    ruta_fichero_tyh):
    f = open(ruta_fichero, "w+")
    g = open(ruta_fichero_data, "w+")
    h = open(ruta_fichero_tyh, "w+")
    return f,g,h
```

- Por último también se han juntado todas las variables comunes en el mismo fichero, para evitar que estuviesen repetidas.

```
electrovalve1 = 'P8_10'
electrovalve2 = 'P8_12'
electrovalve3 = 'P8_14'
electrovalve4 = 'P8_16'

GPIO.setup(electrovalve1,GPIO.OUT)
GPIO.setup(electrovalve2,GPIO.OUT)
GPIO.setup(electrovalve3,GPIO.OUT)
GPIO.setup(electrovalve4,GPIO.OUT)

MAXSUCCION = 100
MINSUCCION = 30
MAXHEAT = 100
MINHEAT = 1
MINSWICHT = 1
MINTENDENCIA = 1
MINTIEMPO = 0
MINSAMPLESINI = 10

# Motor de succion
motorPin = 'P9_21'

sensorTemp22 = DHT.DHT22
Temp22 = 'P8_11'

x=[]     #Almacena el numero de muestra
concentTGS2600=[] #Almacena las muestras de odorante

SLEEP_tyh = 59

odorantes = {1:'Metanol',2:'Etanol',3:'Butanol',4:'Aire'}
electrovalvulas = {1:'ELECTROVALVULA: 1 METANOL \n',
    2:'ELECTROVALVULA: 2 ETANOL \n',3:'ELECTROVALVULA:
    3 BUTANOL \n',4:'ELECTROVALVULA: 4 AIRE \n'}
apertura_electrovalvulas = {1:electrovalve1,2:electrovalve2,
    3:electrovalve3,4:electrovalve4}
```

Todo este código es común a todos los ficheros, y se ha agrupado en el mismo archivo Python, para evitar que esté repetido, y que el código sea más legible.

J

Codigos

J.1. Código principal de PyHuele y del módulo de tratamiento de cadenas

```
# coding: utf-8
#!/usr/bin/python
import re
import numpy as np
import plot as graficas
from datetime import datetime, date
import os
import sys
import puro as puro
import regresion as regresion
import martinelli as martinelli

odorantes = {1:'Metanol',2:'Etanol',3:'Butanol',4:'Aire'}
modulaciones = {1: 'Puro', 2: 'Regresion', 3: 'Martinelli'}
pines = {1: 'P9_38', 2: 'P9_22 y P9_40', 3: 'P9_12 y P9_14'}

MAXSUCCION = 100
MINSUCCION = 30
MAXHEAT = 100
MINHEAT = 1
MINSWICHT = 1
MINTENDENCIA = 1
MINTIEMPO = 0
MINSAMPLESINIO = 10

def get_files_folder(folder,ext=".dat"):

  lstFiles = []
  lstDir = os.walk(folder)

  for root, dirs, files in lstDir:
    for fichero in files:
      (nombreFichero, extension) = os.path.splitext(fichero)
      if(extension == ext):
        lstFiles.append(folder+nombreFichero+extension)
```

```
36
    return lstFiles
38
def crear_transiciones_new(switch):
40    """
      Crea transiciones entre gases
42
      Creamos transiciones entre gases, aleatoriamente, de forma que nunca se repita
        la misma secuencia de apertura.
44    Que no entre dos veces al sensor el mismo gas.

46    Parámetros:
      switch -- numero de gases que van a entrar al sensor
48
      Retorno:
50    devolver -- array con los gases que van a entrar al sensor

52    """
      ret_gases = []
54
      while len(ret_gases) < switch:
56      random = np.random.randint(1,len(odorantes)+1)
        if len(ret_gases) == 0 or random != ret_gases[len(ret_gases)-1]:
58        ret_gases.append([random])

60    return ret_gases

62def todas_transiciones():
      """
64    Crea transiciones entre gases

66    Creamos transiciones entre gases, sin poner aire entre los gases

68    Retorno:
      devolver -- array con los gases que van a entrar al sensor
70
      """
72    return [[1],[1],[2],[2],[3],[3],[4],[4],[2],[1],[3],[1],[4],[3],[2],[4],[1]]

74def tratamiento_elemento(elem,rep):

76    result = []

78    if elem.find("(") != -1:
        aux = re.split("[()]",elem)
80      elem = aux[0]
        rept = 1
82      if len(aux) == 3:
          rept = int(aux[1])
84    elif elem.find("*") != -1:
        elem = re.sub("[*]","",elem)
86      rept = rep
      else:
88      rept = 1

90    try:
        result = [int(e) for e in re.split("[.]*",elem)]
92    except:
        result = [e for e in re.split("[.]*",elem)]
94
      return [result for e in range(rept)],rep-rept
96
def tratamiento_expresion(expr,cont,rep=0):
```

```
98
    result = []
100 while cont < len(expr):

102   if expr[cont].find('{') != -1:
        value = re.sub('{','',expr[cont],count=1)
104
        if value.find('{') == -1:
106       value_aux,rep = tratamiento_elemento(value,rep)
          cont+=1
108     else:
          expr[cont] = value
110       value_aux = []

112     aux_rep = rep
        value,cont,rep,loop,expr,esp_case=tratamiento_expresion(expr,cont,rep)
114     value_aux+=value

116     if esp_case == 1:
          loop = loop/len(value_aux)+1
118       rep-=(loop-1)*len(value_aux)
        else:
120       rep-=(loop-1)*len(value_aux)

122     for j in range(loop):
          result += value_aux
124
      elif expr[cont].find('}') != -1:
126     esp_case = 0
        value = re.split('}',expr[cont],maxsplit=1)
128
        result_aux = []
130     if value[0] != '':
          result_aux,rep = tratamiento_elemento(value[0],rep)
132     result+=result_aux

134     loop = 1
        if value[1].find("}") != -1:
136       if value[1].find("(") == 0:
            value_aux = re.split("[()]",value[1],maxsplit=2)
138
            if value_aux[1] != "":
140           loop = int(value_aux[1])
            expr[cont] = value_aux[2]
142       else:
            expr[cont] = value[1]
144     elif value[1].find("(") != -1:
          value_aux = re.split("[()]",value[1],maxsplit=2)[1]
146       if value_aux != "":
            loop = int(value_aux)
148       cont+=1
        elif value[1].find("*") != -1:
150       loop = rep
          esp_case = 1
152       cont+=1

154     return result,cont,rep,loop,expr,esp_case
      else:
156     result_aux,rep = tratamiento_elemento(expr[cont],rep)
        result+=result_aux
158     cont+=1

160 return result,rep
```

```
162 def flatten(elem):
    if elem == []:
164   return []
    elif isinstance(elem,(int,float,str)):
166   return [elem]
    else:
168   return flatten(elem[0])+flatten(elem[1:])

170 def particionar_elementos(element,rep):

172   data = re.split(",",element)
    data,rep = tratamiento_expresion(data,0,rep)
174   if rep != 0:
      print ("Error al introducir el elemento: " +element+ " revise que todo esté
     bien")
176   exit(1)

178   return flatten(data)

180 def particionar_gases(gases,rep):

182   if rep == 0:
      return gases
184
    gases = re.split(",",gases)
186   response = []

188   for gas in gases:
      if gas.find("*") != -1:
190     gas = re.sub("[*]","",gas)
        response += [tratamiento_expresion(re.split("-",gas),0)[0] for i in range(
     rep)]
192     return response

194   response.append(tratamiento_expresion(re.split("-",gas),0)[0])
      rep-=1
196
    if rep != 0:
198   print ("Error al introducir el elemento: "+element+ " revise que todo esté
     bien")
      exit(1)
200
    return response
202
204 def leer_fichero(path):

206   ret = []
    with open(path) as f:
      for line in f.readlines():
208     line = re.sub('[\s+]','',line)
        line = re.split(':|//',line)
210     ret.append(line)

212   return ret

214 def tratamiento_datos(data):

216   repeticiones = "0*"
    succion = "0*"
218   duracion = "0*"
    switch = "0*"
220   s_ini = "0*"
```

```
    segundosmuestra = "0*"
    nombrearchivo = "0*"
    nombrecarpeta = "0*"
    tendencia = "0*"
    heat = "0*"
    versiones = "0*"
    vecs_open_valves = "0*"
    trans = "0*"
    sleeps = "0*"
    muestras = "0*"
    Tmin = "60*"
    Tmax = "100*"
    martinelli_modo_ejecucion = "1*"
    martinelli_modo_temperatura = "1*"
    mod = 1
    graficar = ""

    for line in data:
        if line[0] == '':
            continue
        elif line[0].lower() == 'numero_experimentos':
            repeticiones = int(line[1])
        elif line[0].lower() == 'succion':
            succion = line[1]
        elif line[0].lower() == 'numero_muestras':
            muestras = line[1]
        elif line[0].lower() == 'duracion_estimulo':
            switch = line[1]
        elif line[0].lower() == 'muestras_iniciales':
            s_ini = line[1]
        elif line[0].lower() == 'segundos_entre_estimulos':
            segundosmuestra = line[1]
        elif line[0].lower() == 'nombre_archivo':
            nombrearchivo = line[1]
        elif line[0].lower() == 'nombre_carpeta':
            nombrecarpeta = line[1]
        elif line[0].lower() == 'tendencia':
            tendencia = line[1]
        elif line[0].lower() == 'calentamiento_sensor':
            heat = line[1]
        elif line[0].lower() == 'version_experimento':
            versiones = line[1]
        elif line[0].lower() == 'vector_apertura_valvulas':
            trans = line[1]
        elif line[0].lower() == 'reposo':
            sleeps = line[1]
        elif line[0].lower() == 'modo_martinelli_ejecucion':
            martinelli_modo_ejecucion = line[1]
        elif line[0].lower() == 'modo_martinelli_temperatura':
            martinelli_modo_temperatura = line[1]
        elif line[0].lower() == 'temperatura_minima_martinelli':
            Tmin = line[1]
        elif line[0].lower() == 'temperatura_maxima_martinelli':
            Tmax = line[1]
        elif line[0].lower() == 'modulacion':
            mod = int(line[1])
        elif line[0].lower() == 'representar':
            graphs = line[1]
        else:
            print ("Error al introducir el identificador de un parámetro. Revise todos
    los nombres. Saliendo")
            exit()

```

```
    succion = particionar_elementos(succion,repeticiones)
    switch = particionar_elementos(switch,repeticiones)
    s_ini = particionar_elementos(s_ini,repeticiones)
    segundosmuestra = particionar_elementos(segundosmuestra,repeticiones)
    nombrearchivo = particionar_elementos(nombrearchivo,repeticiones)
    nombrecarpeta = particionar_elementos(nombrecarpeta,repeticiones)
    versiones = particionar_elementos(versiones,repeticiones)
    muestras = particionar_elementos(muestras,versiones.count(2)) #Con el count(2)
      se obtienen el número de veces que sale el 2
    trans = particionar_gases(trans,versiones.count(3)) #Con el count(3) se
      obtienen el número de veces que sale el 3
    sleeps = particionar_elementos(sleeps,repeticiones)

    tendencia = particionar_elementos(tendencia if mod == 2 else "0*",repeticiones)
    heat = particionar_elementos(heat if mod == 2 else "0*",repeticiones)

    martinelli_modo_ejecucion = particionar_elementos(martinelli_modo_ejecucion if
      mod == 3 else "0*",repeticiones)
    martinelli_modo_temperatura = particionar_elementos(martinelli_modo_temperatura
       if mod == 3 else "0*",repeticiones)
    Tmin = particionar_elementos(Tmin if mod == 3 else "60*",repeticiones)
    Tmax = particionar_elementos(Tmax if mod == 3 else "100*",repeticiones)

    vecs_open_valves_ret = []
    cont = 0
    cont2 = 0

    for v,s in zip(versiones,switch):
      if v == 1:
        vecs_open_valves_ret.append(todas_transiciones())
      elif v == 2:
        vecs_open_valves_ret.append(crear_transiciones_new(muestras[cont2]))
        cont2+=1
      else:
        vecs_open_valves_ret.append(trans[cont])
        cont+=1

    return mod,re.split("[|]",graphs),[succion,switch,s_ini,segundosmuestra,
      nombrearchivo,nombrecarpeta,vecs_open_valves_ret,sleeps,heat,tendencia,
      martinelli_modo_ejecucion,martinelli_modo_temperatura,Tmin,Tmax]

def obtener_ruta(path,ahora,folder,samplesinicio,folder2):
    return path + str(ahora.year)+str(ahora.month)+str(ahora.day) + "/" + folder +
      "/" +  str(samplesinicio) + "SAMPLESINICIO" + "/" + folder2 + "/"

def sort_files(file,name_files,folders,path,ahora):

    for folder in folders:
      for name_file in name_files:
        try:
          return datetime.strptime(file,path+str(ahora.year)+str(ahora.month)+str(
      ahora.day)+'/'+folder[1]+'/'+str(folder[0])+'SAMPLESINICIO/dat/'+name_file+'
      _dat1_%a%d%b%Y-%H%M%S.dat')
        except:
          continue

def main():

    data = leer_fichero(sys.argv[1])
    mod,graphs,data = tratamiento_datos(data)

    graphs.pop()
    graphs = [int(graph) for graph in graphs]
```

```
338  for suc,sw,sini,tend,he in zip(data[0],data[1],data[2],data[8],data[9]):
       if suc > MAXSUCCION or suc < MINSUCCION or sw < MINSWICHT or (mod != 3 and
       sini < MINSAMPLESINIO) or (mod == 2 and (he < MINHEAT or he > MAXHEAT or tend
       < MINTENDENCIA)):
340      print ('PARÁMETROS(S) INCORRECTO(S).\n Por favor revise los parámetros
       introducidos y vuelva a empezar')
         exit(1)
342
     ahora = datetime.now()
344
     if mod == 1: #Mensajes de error
346    puro.puro_captura_datos(ahora,data[0:8])
       path = "CAPTURAS/PURO/"
348  elif mod == 2:
       regresion.regresion_captura_datos(ahora,data[0:10])
350    path = "CAPTURAS/REGRESION/"
     else:
352    martinelli.martinelli_captura_datos(ahora,data[0:8]+data[10:])
       path = "CAPTURAS/MARTINELLI/"
354
     folders = []
356  [folders.append((sini,foldename)) for sini,foldename in zip(data[2],data[5]) if
       (sini,foldename) not in folders] #Obtengo las carpetas donde estan los
       ficheros con los datos

358  name_files = []
     [name_files.append(name) for name in data[4] if name not in name_files] #
       Obtengo los nombres introducidos
360
     files = flatten([get_files_folder(obtener_ruta(path,ahora,f_name[1],f_name[0],"
       dat")) for f_name in folders]) #Ya tenemos los archivos en una lista
362  files.sort()

364  dictionary = {}
     for swt,sini,CTE,folder,tran,texecution,mexecution,Tmin,Tmax,file in zip(data
       [1],data[2],data[3],data[5],data[6],data[10],data[11],data[12],data[13],files)
       :
366    if dictionary.has_key((swt,sini,CTE,len(tran),texecution,mexecution,Tmin,Tmax
       ,folder)) == False:
         dictionary[(swt,sini,CTE,len(tran),texecution,mexecution,Tmin,Tmax,folder)]
       = [file]
368    else:
         list_files = dictionary.get((swt,sini,CTE,len(tran),texecution,mexecution,
       Tmin,Tmax,folder))
370      list_files.append(file)
         dictionary[(swt,sini,CTE,len(tran),texecution,mexecution,Tmin,Tmax,folder)]
       = list_files
372
     for graph in graphs:
374    if mod != 3:
         if graph == 1:
376        graficas.plot_graf(dictionary,ahora,path) #data[1] = switch, data[2] =
       s_ini, data[3] = CTE, data[6] = tran
         elif graph == 2:
378        graficas.plot_elem_grafs(dictionary,ahora,path)
         else:
380        graficas.plot_elems_div_graf(dictionary,ahora,path)
       else:
382      if graph == 1:
           graficas. plot_martinelli(dictionary,ahora,path)
384      elif graph == 2:
           graficas.plot_martinelli_time_pulses(dictionary,ahora,path)
```

```
386         else:
              graficas.plot_martinelli_div_graf(dictionary,ahora,path)
388
if __name__ == '__main__':
390   main()
```

codigos/PyHuele.py

J.2. Código del módulo de captura de datos

J.2.1. Algoritmo de temperatura constante

```
# -*- encoding: utf-8 -*- # Especificamos que nuestro archivo .py estÃą
    codificado en UTF-8
2 #!/usr/bin/python


4
import EN_codigo_comun as ENC
6 import Adafruit_BBIO.PWM as PWM
import Adafruit_BBIO.ADC as ADC
8 import time
from datetime import datetime, date
10 from threading import Thread


12
sensorPin2600 = 'P9_38'   #Sensor TGS2600 (Puerto AIN_3)
14
SAMPLESINICIO=None     #Capturas iniciales
16
# Sensor TGS2600.
18
Rl_2600=440 #Omh
20 Vc=5 #V

22 f = None
g = None
24 h = None

26 t = None

28 def puro_TGS2600(count,string,gas):
  """
30   Captura muestras sin usar ninguna técnica.

32   Realiza la toma de medidas sin usar ninguna técnica de modulación

34   Parámetros:
   count -- contador que indica las muestras que hemos tomado
36   string -- cadena que indica si una muestra es inicial o no, se usa al imprimir
       en los ficheros
   gas -- array que contiene los gases de la muestra que se captura
38
   """
40
   value=0
42   queda=0
   residuo=0
44   ADC.read_raw(sensorPin2600) #la primera medida es erronea por el bug

46   for i in range(ENC.NM):
```

```
        value += ADC.read_raw(sensorPin2600)
48      time.sleep(ENC.tsub)


50
    valueTGS2600=value/ENC.NM
52  if valueTGS2600 >= 1799.0:
        print "El valor supera los 1'8V y puede estropear la BBB, se sale de la
        ejecucion"
54      f.write("El valor supera los 1'8V y puede estropear la BBB, se sale de la
        ejecucion")
        f.flush()
56      PWM.stop("P9_21")
        PWM.cleanup()
58      ENC.cierre(f,g,h,t)
        exit()
60  instante_captura=datetime.now()

62  ENC.concentTGS2600.append(valueTGS2600)

64  #Se calcula el valor de la resistencia interna del sensor
    RsTGS2600=((Vc*Rl_2600)/(valueTGS2600/1000.))-Rl_2600
66
    #Escritura por pantalla de la lectura
68  print string+'['+str(count)+']\n>> Valor: '+str(valueTGS2600)+'mV >> Rs: '+str(
      RsTGS2600)+' >> Temperatura: 100 >> '+str(instante_captura)

70  #Se escriben los datos en el fichero
    wline_g=str(count)+' '+str(valueTGS2600)+' '+str(RsTGS2600)+' 100 '+str(
      instante_captura)
72  for elem in gas:
        wline_g+=' '+ str(elem) +' '+ ENC.odorantes[elem]
74  wline_g+='\n'
    wline_f=string+'['+str(count)+']  Valor:'+str(valueTGS2600)+'mV -- Rs: '+str(
      RsTGS2600)+' --Temperatura:  100 >>'+str(instante_captura)+' Gas(Gases)
      captados e identificadores'
76  for elem in gas:
        wline_f+=' '+ str(elem) +' '+ ENC.odorantes[elem]
78  wline_f+='\n'
    g.writelines(wline_g)
80  g.flush()
    f.writelines(wline_f)
82  f.flush()

84 def captura_odorante(vector_odorantes,inicio,fin,string):
    """
86  Capturamos tantas muestras como se indique en los argumentos

88  Codigo común para la captura de gases, que llama a la función de puro_TGS2600,
      abre y cierra las válvulas
    y mide el tiempo de captura de los gases.
90
    Parámetros:
92  vector_odorantes -- el vector de los odorantes que van a captarse en esta
      apertura de válvulas
    inicio -- el número que marca el inicio de muestras que hay que coger
94  fin -- el número final de muestras que hay que coger
    string -- cadena que indica si una muestra es inicial o no, se usa al imprimir
      en los ficheros
96
    """
98
    ENC.abrir_electrovalvulas(vector_odorantes)
100 for count in range(inicio,fin):
```

```
    time_ini = time.time()
102     puro_TGS2600(count,string,vector_odorantes)
    time_end = time.time()
104     time.sleep(ENC.SLEEP - (time_end - time_ini))
  ENC.cerrar_electrovalvulas()
106
def puro_captura_datos(ahora,data):
108
  global SAMPLESINICIO,t
110   succion,switch,samples_ini,CTE,name_file,name_folder,vecs_open_valves,sleeps =
    data

112   for suc,sw,SAMPLESINICIO,CT,nfile,nfolder,vsovs,sle in zip(succion,switch,
    samples_ini,CTE,name_file,name_folder,vecs_open_valves,sleeps):

114     print '\nSensor: TGS2600     Pin ADC: ' +sensorPin2600
    print 'Algoritmo: PURO SIN MODULACION'
116     ti2 = float(SAMPLESINICIO + (CT*(len(vsovs)+1)) + (len(vsovs)*sw))/60
    print 'Succion motor al ' +str(suc)+ '%    motor Pin PWM : ' +ENC.motorPin
118     print 'Tiempo conmutacion electrovalvula: ' +str(sw)+' segundos'
    print 'Tiempo de experimentación: ' +str(ti2)+ ' minutos\n'
120     print 'Muestras iniciales de la experimentación: ' +str(SAMPLESINICIO)+'\n'
    print 'Cantidad de tiempo en segundos entre gases: ' +str(CT)+'\n'
122     print 'Se ha escogido el siguiente nombre para los ficheros: '+nfile+'\n'
    print 'Se ha escogido el siguiente nombre para la carpeta: '+nfolder+'\n'
124     print 'Conmutacion entre electrovalvulas:  ' +str(vsovs)+ '\n'

126     ruta_fichero,ruta_fichero_data,ruta_fichero_tyh = ENC.create_files(nfile,"
    CAPTURAS/PURO",nfolder,SAMPLESINICIO,ahora)
    global f,g,h
128     f,g,h = ENC.open_files(ruta_fichero,ruta_fichero_data,ruta_fichero_tyh)

130     print 'Ruta Fichero: ' + ruta_fichero
    print 'Ruta Fichero de datos: '+ ruta_fichero_data
132
    #Llamada al thread de medida de temperatura y humedad
134     t = Thread(target=ENC.measure_tyh,args=(h,))
    t.do_run = True
136     t.start()
    ADC.setup()
138
    ENC.file_TGS2600(vsovs,suc,100,0,"5V",ti2,sw,"Puro","CAPTURAS/PURO/"+ str(
    ahora.year)+str(ahora.month)+str(ahora.day) + "/"+nfolder + "/" + str(
    SAMPLESINICIO) + "SAMPLESINICIO/",nfile+".txt",nfile+".dat","","TyH"+nfile+".
    data",f,1,CT,SAMPLESINICIO)
140     PWM.start("P9_21",suc)

142     #Calentamiento del sensor, se toman SAMPLESINICIO medidas antes de comenzar
    la experimentacion
    print '\n\nComienza la adquisicion. Muestras inciales: ' +str(SAMPLESINICIO)+
     '\n\n'
144     f.write('\n\nComienza la adquisicion. Muestras inciales: ' +str(SAMPLESINICIO
    )+ '\n\n')
    f.flush()
146     ENC.cerrar_electrovalvulas()
    captura_odorante([4],0,SAMPLESINICIO,"Muestra_ini")
148
    print '\n\nComienza la experimentacion\n'
150     f.write('\n\nComienza la experimentacion\n')
    f.flush()
152
    #Comienza la experimentacion
154
```

```
    ENC.cerrar_electrovalvulas()
156 for i in range(len(vsovs)):

158   if CT > 0:
        captura_odorante([4],len(ENC.concentTGS2600),(len(ENC.concentTGS2600)+CT)
      ,"Muestra_bucle")
160
      ENC.imprimir_electrovalvulas(vsovs[i])
162   captura_odorante(vsovs[i],len(ENC.concentTGS2600),int(len(ENC.
    concentTGS2600)+sw),"Muestra")

164 if CT > 0:
      captura_odorante([4],len(ENC.concentTGS2600),(len(ENC.concentTGS2600)+CT),"
    Muestra_bucle")
166
    PWM.stop("P9_21")
168 PWM.cleanup()
    ENC.cierre(f,g,h,t)
170 ENC.x=[]
    ENC.concentTGS2600=[]
172 time.sleep(sle)

174 return 0
```

codigos/puro.py

J.2.2. Algoritmo de temperatura variable y codificación en amplitud

```
  # coding: utf-8
2


4 import EN_codigo_comun as ENC
  import Adafruit_BBIO.PWM as PWM
6 import Adafruit_BBIO.ADC as ADC
  import time
8 from scipy import stats
  from datetime import datetime, date
10 from threading import Thread

12 sensorPin2600 = 'P9_40'
  heatPin2600 = 'P9_22'
14 Vc = 5 #V

16 f = None
  g = None
18 h = None

20 t = None

22 SAMPLESINICIO = None
  TENDENCIA = None
24


26 tempTGS2600 = []
  Rl_2600 = 440 # Omh
28 resTGS2600 = []

30 def regresion_TGS2600(count, temperature_TGS2600,string,opcion,gas):
     """
32   Captura muestras usando la técnica de la regresión lineal
```

```
    Realiza la toma de medidas usando como técnica de modulación la regresion lineal
    de la temperatura en función de la muestra obtenida en el sensor.

    Parámetros:
    count -- contador que indica las muestras que hemos tomado
    temperature_TGS2600 -- temperatura inicial a la que calentamos el sensor
    string -- cadena que indica si una muestra es inicial o no, se usa al imprimir en los ficheros
    opcion -- 1 para las SAMPLESINICIO muestras, 2 para capturas tantas muestras como hayamos indicado
    gas -- array que contiene los gases de la muestra que se captura

    """

    if opcion == 1:
      PWM.set_duty_cycle(heatPin2600, temperature_TGS2600)

    value=0
    queda=0
    residuo=0
    ADC.read_raw(sensorPin2600) #la primera medida es erronea por el bug

    for i in range(ENC.NM):
       value += ADC.read_raw(sensorPin2600)
       time.sleep(ENC.tsub)

    valueTGS2600=value/ENC.NM
    if valueTGS2600 >= 1799.0:
       print "El valor supera los 1'8V y puede estropear la BBB, se sale de la ejecucion"
       f.writelines("El valor supera los 1'8V y puede estropear la BBB, se sale de la ejecucion")
       f.flush()
       PWM.stop('P9_21')
       PWM.stop(heatPin2600)
       PWM.cleanup()
       ENC.cierre(f,g,h,t)
       exit()
    instante_captura=datetime.now()

    #Adaptacion temperatura
    if opcion == 2:
       slope, intercept, r_value, p_value, std_err1 = stats.linregress(ENC.x[(count-SAMPLESINICIO):(count-1)],
          ENC.concentTGS2600[(count-SAMPLESINICIO):(count-1)])

       temperature_TGS2600 = temperature_TGS2600 - (slope*TENDENCIA)

       if temperature_TGS2600 < 10.0:
          temperature_TGS2600 = 10.0
       elif temperature_TGS2600 > 90.0:
          temperature_TGS2600 = 90.0

       print "Los valores de la tendencia, el slope y la temperatura son: "+str(TENDENCIA)+", "+str(slope)+" y "+str(temperature_TGS2600)
       #Reset de setup PWM
       PWM.set_duty_cycle(heatPin2600, temperature_TGS2600)

    #Se calcula el valor de la resistencia interna del sensor
    RsTGS2600=((Vc*Rl_2600)/(valueTGS2600/1000.))-Rl_2600

    ENC.x.append(count)
```

```
90    ENC.concentTGS2600.append(valueTGS2600)
      tempTGS2600.append(temperature_TGS2600)
92    resTGS2600.append(RsTGS2600)

94    #Escritura por pantalla de la lectura
      print string+'['+str(count)+']\n>> Valor: '+str(valueTGS2600)+'mV >> Rs: '+str(
        RsTGS2600)+' >> Temperatura:'+str(temperature_TGS2600)+' >> '+str(
        instante_captura)+'\n'
96
      #Se escriben los datos en el fichero
98    wline_g=str(count)+' '+str(valueTGS2600)+' '+str(RsTGS2600)+' '+str(
        temperature_TGS2600)+' '+str(instante_captura)
      for elem in gas:
100     wline_g+=' '+ str(elem) +' '+ ENC.odorantes[elem]
      wline_g+='\n'
102   wline_f=string+'['+str(count)+']  Valor:'+str(valueTGS2600)+'mV -- Rs: '+str(
        RsTGS2600)+' --Temperatura: '+str(temperature_TGS2600)+' >>'+str(
        instante_captura)+' Gas(Gases) captados e identificadores:'
      for elem in gas:
104     wline_f+=' '+ str(elem) +' '+ ENC.odorantes[elem]
      wline_f+='\n'
106   g.write(wline_g)
      g.flush()
108   f.write(wline_f)
      f.flush()
110
  def captura_odorante(vector_odorantes,inicio,fin,string,opcion,heat2600):
112
      """
114   Capturamos tantas muestras como se indique en los argumentos

116   Codigo común para la captura de gases, que llama a la función de puro_TGS2600,
        abre y cierra las válvulas
      y mide el tiempo de captura de los gases.
118
      Parámetros:
120   vector_odorantes -- el vector de los odorantes que van a captarse en esta
        apertura de válvulas
      inicio -- el número que marca el inicio de muestras que hay que coger
122   fin -- el número final de muestras que hay que coger
      string -- cadena que indica si una muestra es inicial o no, se usa al imprimir
        en los ficheros
124   opcion -- 1 para las SAMPLESINICIO muestras, 2 para capturas tantas muestras
        como hayamos indicado
      heat2600 -- temperatura inicial a la que calentamos el sensor
126
      """
128
      ENC.abrir_electrovalvulas(vector_odorantes)
130   for count in range(inicio,fin):
        time_ini = time.time()
132     regresion_TGS2600(count, heat2600,string,opcion,vector_odorantes)
        time_end = time.time()
134     time.sleep(ENC.SLEEP - (time_end - time_ini))
      ENC.cerrar_electrovalvulas()
136
  def regresion_captura_datos(ahora,data):
138



140
      global TENDENCIA,SAMPLESINICIO,t
142   succion,switch,samples_ini,CTE,name_file,name_folder,vecs_open_valves,sleeps,
        heat,tendencia = data
```

```
144  for suc,he,sw,TENDENCIA,SAMPLESINICIO,CT,nfile,nfolder,vsovs,sle in zip(succion
       ,heat,switch,tendencia,samples_ini,CTE,name_file,name_folder,vecs_open_valves,
       sleeps):

146    print '\nSensor: TGS2600    Pin ADC: ' + sensorPin2600
       print 'Algoritmo: REGRESION TEMPERATURA'
148    print suc,he,sw,TENDENCIA,SAMPLESINICIO,CT,nfile,nfolder,vsovs,sle
       ti2= float(SAMPLESINICIO + (CT*(len(vsovs)+1)) + (len(vsovs)*sw))/60
150    print 'Succion motor al ' + str(suc) + '%    motor Pin PWM : ' + ENC.motorPin
       + '\n'
       print 'Calentamiento promedio: ' + str(he) + '%    TGS2600 Pin PWM : ' +
       heatPin2600 + '\n'
152    print 'Tiempo de experimentaciÃşn: ' + str(ti2) + ' minuto(s) \n'
       print 'Tiempo conmutacion electrovalvula: ' +str(sw)+' segundos\n'
154    print 'Tendencia: ' +str(TENDENCIA)+'\n'
       print 'Muestras iniciales de la experimentación: ' +str(SAMPLESINICIO)+'\n'
156    print 'Cantidad de tiempo en segundos entre gases: ' +str(CT)+'\n'
       print 'Se ha escogido el siguiente nombre para los ficheros: '+nfile+'\n'
158    print 'Se ha escogido el siguiente nombre para la carpeta: '+nfolder+'\n'
       print 'Conmutacion entre electrovalvulas:  ' + str(vsovs) + '\n'
160
       ruta_fichero,ruta_fichero_data,ruta_fichero_tyh = ENC.create_files(nfile,"
       CAPTURAS/REGRESION",nfolder,SAMPLESINICIO,ahora)
162    global f,g,h
       f,g,h = ENC.open_files(ruta_fichero,ruta_fichero_data,ruta_fichero_tyh)
164
       print 'Ruta Fichero: ' + ruta_fichero
166    print 'Ruta Fichero de datos: '+ ruta_fichero_data

168    #Llamada al thread de medida de temperatura y humedad
       t = Thread(target=ENC.measure_tyh,args=(h,))
170    t.do_run = True
       t.start()
172    PWM.start("P9_21",suc)
       ADC.setup()
174    PWM.start(heatPin2600,he,20000,0)

176    ENC.file_TGS2600(vsovs,suc,he,TENDENCIA,heatPin2600,ti2,sw,"Regresion","
       CAPTURAS/REGRESION/"+ str(ahora.year)+str(ahora.month)+str(ahora.day) + "/"+
       nfolder + "/"  + str(SAMPLESINICIO) + "SAMPLESINICIO/",nfile+".txt",nfile+".
       dat","","TyH"+nfile+".data",f,1,CT,SAMPLESINICIO)

178    print '\n\nComienza la adquisicion. Muestras inciales: ' +str(SAMPLESINICIO)+
        '\n\n'
       f.write('\n\nComienza la adquisicion. Muestras inciales: ' +str(SAMPLESINICIO
       )+ '\n\n')
180    f.flush()
       ENC.cerrar_electrovalvulas()
182
       #Calentamiento del sensor, se toman SAMPLESINICIO medidas antes de comenzar
       la experimentacion
184    captura_odorante([4],0,SAMPLESINICIO,"Muestra Regresion ini",1,he)

186    print '\n\nComienza la experimentacion\n'
       f.write('\n\nComienza la experimentacion\n')
188    f.flush()

190    for i in range(len(vsovs)):

192      if CT > 0:
           captura_odorante([4],len(ENC.concentTGS2600),(len(ENC.concentTGS2600)+CT)
       ,"Muestra Regresion Bucle",2,he)
```

```
194
        ENC.imprimir_electrovalvulas(vsovs[i])
196     captura_odorante(vsovs[i],len(ENC.concentTGS2600),int(len(ENC.
    concentTGS2600)+sw),"Muestra Regresion",2,he)

198     if CT > 0:
          captura_odorante([4],len(ENC.concentTGS2600),(len(ENC.concentTGS2600)+CT),"
    Muestra Regresion Bucle",2,he)
200
    PWM.stop('P9_21')
202 PWM.stop(heatPin2600)
    PWM.cleanup()
204 ENC.cierre(f,g,h,t)
    ENC.x=[]
206 ENC.concentTGS2600=[]
    time.sleep(sle)
208
  return 0
```

codigos/regresion.py

J.2.3. Algoritmo de temperatura variable y codificación en frecuencia

```
# coding: utf-8
2
import EN_codigo_comun as ENC
4 import Adafruit_BBIO.ADC as ADC
import Adafruit_BBIO.GPIO as GPIO
6 import Adafruit_BBIO.PWM as PWM
import time
8 from datetime import datetime, date
from threading import Thread
10 import os


12
sensorPin555 = 'P9_12'
14 heatPin2600 = 'P9_14'
GPIO.setup(sensorPin555,GPIO.IN)
16 SAMPLESINICIO=None
heat2600=100
18
x=[]
20 up=[]
down=[]
22 tempTGS2600=[]
duracion=[]
24
concentTGS2600=[]
26 Rl_2600=27000
Vc=5
28
f = None
30 g = None
h = None
32 k = None

34 t = None

36 def adjust_temperature_TGS(mode_temperature,periodo,subperiodo,Tmin,Tmax):

38   if mode_temperature == 1:
```

```
    if periodo == 0:
      return Tmin
    else:
      return Tmax
  else:
    if periodo == 0:
      return Tmax - (((Tmax-Tmin)/8)*(subperiodo+1))
    else:
      return Tmin + (((Tmax-Tmin)/8)*(subperiodo+1))

def martinelli(stringA, stringB, gas, mode_temperature, Tmin, Tmax):
  """
  Captura muestras usando la técnica de Martinelli

  Realiza la toma de medidas usando como técnica la descrita por Martinelli

  Parámetros:
  stringA -- cadena que indica si una muestra es inicial o no, se usa al imprimir en los ficheros
  stringB -- cadena que indica si una muestra es inicial o no, se usa al imprimir en los ficheros
  gas -- array que contiene los gases de la muestra que se captura

  """

  count = 0
  sumatorio = 0
  value_down = GPIO.wait_for_edge(sensorPin555, GPIO.FALLING) #Espera pulso de caida

  #Realizamos dos tomas de 8 cada una,16 en total, variando unicamente la temperatura
  for i in range(2):
    for j in range(8):

      temperature_TGS2600 = adjust_temperature_TGS(mode_temperature,i,j,Tmin,Tmax)
      #Fijamos la temperatura
      PWM.set_duty_cycle(heatPin2600, temperature_TGS2600)
      #Espera pulso de caida
      instante_captura_ini=datetime.now() #Lo pongo afuera para que no interfiera en la captura.
      value_down = GPIO.wait_for_edge(sensorPin555, GPIO.FALLING)
      ini_pulso = time.time()

      value_up = GPIO.wait_for_edge(sensorPin555, GPIO.RISING)
      ini_up = time.time()

      value_down = GPIO.wait_for_edge(sensorPin555, GPIO.FALLING)   #Espera pulso de bajada
      fin_pulso = time.time()
      instante_captura=datetime.now()

      time_down = ini_up - ini_pulso
      time_up = fin_pulso - ini_up    #Duracion del pulso en estado alto
      time_pulso = fin_pulso - ini_pulso  #Duracion del pulso completo

      x.append(count)
      up.append(time_up)
      down.append(time_down)
      duracion.append(time_pulso)
      sumatorio+=time_pulso

```

```
      print 'Muestra '+stringA+'['+str(count)+']\n>> t_up='+str(time_up)+'  >>
    t_down: '+str(time_down)+ '  >> Duracion pulso: '+str(time_pulso)
96    print '>> Heat: '+str(temperature_TGS2600)+'  >> '+str(instante_captura)+'\
    n'

98    #Se escriben los datos en el fichero
      wline_g=str(count)+'  '+str(time_up)+'  '+str(time_down)+'  '+str(
    time_pulso)+' '+str(temperature_TGS2600)+' '+str(instante_captura_ini)+'  '+
    str(instante_captura)
100   for elem in gas:
        wline_g+=' '+ str(elem) +' '+ ENC.odorantes[elem]
102   wline_g+='\n'
      wline_f= stringB+'['+str(count)+' Heat: '+str(temperature_TGS2600)+'] t_up=
    '+str(time_up)+' t_down: '+str(time_down)+'  Instante captura inicial: '+str(
    instante_captura_ini)+ '   Durancion pulso: '+str(time_pulso)+'>> '+str(
    instante_captura)+'\n'
104   g.writelines(wline_g)
      g.flush()
106   f.writelines(wline_f)
      f.flush()
108   count+=1

110 return sumatorio

112 def captura_odorante(stringA,stringB,stringC,i,vector_odorantes,mode_execution,
    mode_temperature,Tmin,Tmax):

114 """
  Capturamos los 16 pulsos del gas.
116
  Codigo común para la captura de gases, que llama a la función de martinelli,
    abre y cierra las válvulas
118 y mide el tiempo de captura de los gases.

120 Parámetros:
  stringA -- cadena que indica si una muestra es inicial o no, se usa al imprimir
     en los ficheros
122 stringB -- cadena que indica si una muestra es inicial o no, se usa al imprimir
     en los ficheros
  stringC -- cadena que indica si una muestra es inicial o no, se usa al imprimir
     en los ficheros
124 i -- contador que muestra cuantos veces hemos capturado muestras
  vector_odorantes -- el vector de los odorantes que van a captarse en esta
    apertura de válvulas
126
  """
128
  time_martinelli = martinelli(stringA,stringB,vector_odorantes,mode_temperature,
    Tmin,Tmax)
130 instante_captura = datetime.now()
  print'\n'+stringC+str(i)+'] Duracion de los k=16 pulsos:'+str(time_martinelli)+
    ' >> '+str(instante_captura)+'\n'
132 wline_f = stringC+str(i)+'] Duracion de los k=16 pulsos:'+str(time_martinelli)+
    ' >> '+str(instante_captura)+'\n'
  wline_k = str(i) + ' ' + str(time_martinelli) + ' ' + str(instante_captura) + '
    \n'
134 f.writelines(wline_f)
  f.flush()
136 k.writelines(wline_k)
  k.flush()
138
  if mode_execution == 1:
140   return time_martinelli
```

```
    else:
142     return 1

144 def martinelli_captura_datos(ahora,data):


146
    global SAMPLESINICIO,t
148 succion,conmutacion,samples_ini,CTE,name_file,name_folder,vecs_open_valves,
      sleeps,martinelli_execution_modes,martinelli_temperature_modes,Tmins,Tmaxs =
      data

150 for suc,con,SAMPLESINICIO,CT,nfile,nfolder,vsovs,sle,mode_execution,
      mode_temperature,Tmin,Tmax in zip(succion,conmutacion,samples_ini,CTE,
      name_file,name_folder,vecs_open_valves,sleeps,martinelli_execution_modes,
      martinelli_temperature_modes,Tmins,Tmaxs):

152     print 'Algoritmo: MARTINELLI'
        print 'Succion motor al '+str(suc)+'%    Motor Pin PWM: '+ENC.motorPin
154     print 'Duracion de la experimentacion '+str(len(vsovs))+' muestras'
        print 'Tiempo de apertura de cada electrovalvula '+str(con)+' segundos\n'
156     print 'Muestras iniciales de la experimentación: '+str(SAMPLESINICIO)+'\n'
        print 'Cantidad de tiempo en segundos entre gases: '+str(CT)+'\n'
158     print 'Se ha escogido el siguiente nombre para los ficheros: '+nfile+'\n'
        print 'Se ha escogido el siguiente nombre para la carpeta: '+nfolder+'\n'
160     print 'Conmutacion entre electrovalvulas: '+str(vsovs)+'\n'

162     ruta_fichero,ruta_fichero_data1,ruta_fichero_tyh = ENC.create_files(nfile,"
      CAPTURAS/MARTINELLI",nfolder,SAMPLESINICIO,ahora)
        ruta_carpeta_2 = ("CAPTURAS/MARTINELLI/" + str(ahora.year)+str(ahora.month)+
        str(ahora.day) + "/" + nfolder + "/" + str(SAMPLESINICIO) + "SAMPLESINICIO" +
        "/" + "dat2" + "/")
164     if not os.path.exists(ruta_carpeta_2): os.makedirs(ruta_carpeta_2)  #Si la
      ruta no existe, se crea
        ruta_fichero_data2 = ruta_carpeta_2.strip() + str(nfile+'_dat2_'+(time.
        strftime("%a%d%b%Y-%HH%MM%SS",time.localtime())+".dat"))
166


168     global f,g,h,k
        f,g,h = ENC.open_files(ruta_fichero,ruta_fichero_data1,ruta_fichero_tyh)
170     k = file(ruta_fichero_data2, "w")

172     print 'Ruta Fichero: '+ruta_fichero
        print 'Ruta Fichero de datos 1: '+ruta_fichero_data1
174     print 'Ruta Fichero de datos 2: '+ruta_fichero_data2

176     #Llamada al thread de medida de temperatura y humedad
        t = Thread(target=ENC.measure_tyh,args=(h,))
178     t.do_run = True
        t.start()
180
        #Se realiza el Setup de los puertos ADC, PWM, GPIO
182     PWM.start(heatPin2600,heat2600,20000,0)
        PWM.start("P9_21",suc)
184     ADC.setup()

186     ENC.file_TGS2600(vsovs,suc,heat2600,0,heatPin2600,len(vsovs),con,"Martinelli"
        ,"/home/enangel/huele/CAPTURAS/MARTINELLI/"+ str(ahora.year)+str(ahora.month)+
        str(ahora.day) + "/"+nfolder + "/" + str(SAMPLESINICIO) + "SAMPLESINICIO/",
        nfile+".txt",nfile+".dat",nfile+"2.dat","TyH"+nfile+".dat",f,2,CT,
        SAMPLESINICIO)

188     print '\nComienza la adquisicion. Muestras inciales: '+str(SAMPLESINICIO)+'
        \n'
```

```
    f.write('\n\nComienza la adquisicion. Muestras inciales: ' +str(SAMPLESINICIO
    )+ '\n')
190 f.flush()
    ENC.cerrar_electrovalvulas()
192
    #TOMAMOS X MUESTRAS DE INICIO
194 ENC.abrir_electrovalvulas([4])
    for i in range(SAMPLESINICIO):
196   captura_odorante('Martinelli_ini','Muestra_ini','Samples inicio_Iteracion['
    ,i,[4],mode_execution,mode_temperature,Tmin,Tmax)
    ENC.cerrar_electrovalvulas()
198
    print '\n\nComienza la experimentacion\n'
200 f.write('\n\nComienza la experimentacion\n')
    f.flush()
202
    w = 0
204 for i in range(len(vsovs)):

206   if CT > 0:
        tiempo = 0
208     ENC.abrir_electrovalvulas([4])
        while tiempo < CT:
210       tiempo+=captura_odorante('Martinelli_bucle', 'Muestra_bucle','Iteracion
     bucle [',w,[4],mode_execution,mode_temperature,Tmin,Tmax)
        ENC.cerrar_electrovalvulas()
212     w+=1

214   ENC.imprimir_electrovalvulas(vsovs[i])
      ENC.abrir_electrovalvulas(vsovs[i])
216
      tiempo_valve = 0
218   while tiempo_valve < con: # Capturas tanto tiempo como conmutacion desees
        tiempo_valve +=captura_odorante('Martinelli', 'Muestra','Iteracion [',w,
    vsovs[i],mode_execution,mode_temperature,Tmin,Tmax)
220     print 'Tiempo acumulado = '+str(tiempo_valve)
      ENC.cerrar_electrovalvulas()
222   w+=1

224 if CT > 0:
      tiempo = 0
226   ENC.abrir_electrovalvulas([4])
      while tiempo < CT:
228     tiempo+=captura_odorante('Martinelli_bucle', 'Muestra_bucle','Iteracion
    bucle [',w,[4],mode_execution,mode_temperature,Tmin,Tmax)
      ENC.cerrar_electrovalvulas()
230   w+=1

232 PWM.stop('P9_21')
    PWM.stop(heatPin2600)
234 PWM.cleanup()
    ENC.cierre(f,g,h,t)
236 k.close()
    ENC.x=[]
238 ENC.concentTGS2600=[]
    time.sleep(sle)
240
  return 0
```

codigos/martinelli.py

J.2.4. Código del común de las distintas técnicas para la captura de odorantes

```
# coding: utf-8

import Adafruit_BBIO.GPIO as GPIO
import Adafruit_DHT as DHT
import time
import os
from datetime import datetime, date
from threading import currentThread

# ASIGNACION DE PUERTOS.

electrovalve1 = 'P8_10'
electrovalve2 = 'P8_12'
electrovalve3 = 'P8_14'
electrovalve4 = 'P8_16'

GPIO.setup(electrovalve1,GPIO.OUT)
GPIO.setup(electrovalve2,GPIO.OUT)
GPIO.setup(electrovalve3,GPIO.OUT)
GPIO.setup(electrovalve4,GPIO.OUT)

MAXSUCCION = 100
MINSUCCION = 30
MAXHEAT = 100
MINHEAT = 1
MINSWICHT = 1
MINTENDENCIA = 1
MINTIEMPO = 0
MINSAMPLESINI = 10

# Motor de succion
motorPin = 'P9_21'

sensorTemp22 = DHT.AM2302#DHT.DHT22
# Pin de lectura de temperatura/Humedad con DHT22 (Puerto GPIO_45)
Temp22 = 'P8_11'

NM=10 #Numero lecturas ADC
T=0.1 #Las NM lecturas se hacen en T segundos
tsub=T/NM; #Subdivisiones de tiempo para las lecturas ADC
SLEEP=1 #Periodo de estabilizacion cuando se pone una nueva temperatura

x=[]     #Almacena el numero de muestra
concentTGS2600=[] #Almacena las muestras de odorante

# Measure temperatura y humedad
SLEEP_tyh = 59

odorantes = {1:'Metanol',2:'Etanol',3:'Butanol',4:'Aire'}
odorantes2 = {'metanol':1,'Metanol':1,'etanol':2,'Etanol':2,'butanol':3,'Butanol'
    :3,'aire':4,'Aire':4}
electrovalvulas = {1:'ELECTROVALVULA: 1 METANOL \n',2:'ELECTROVALVULA: 2 ETANOL \
    n',3:'ELECTROVALVULA: 3 BUTANOL \n',4:'ELECTROVALVULA: 4 AIRE \n'}
apertura_electrovalvulas = {1:electrovalve1,2:electrovalve2,3:electrovalve3,4:
    electrovalve4}
entrada = {0:'No',1:'Si'}

def create_files(nameFile,path,folder,samplesinicio,ahora):
  """
  Crea los ficheros para escribir
```

```
59    Crea los ficheros necesarios para guardar todos los datos de las
      experimentaciones.
61
      Parámetros:
63    nameFile -- nombre que va a usarse para crear los ficheros que contendran los
        datos y
      el resto de la información
65    path -- ruta donde se va a guardar el fichero
      folder -- nombre de la carpeta donde va a guardarse el archivo
67
      Retorno:
69    devolver -- array con los gases que van a entrar al sensor

71    """
      fileString = nameFile+"_txt_" +time.strftime("%a%d%b%Y-%HH%MM%SS", time.
        localtime())+".txt"
73    fileString_data = nameFile+"_dat1_"+time.strftime("%a%d%b%Y-%HH%MM%SS", time.
        localtime())+".dat"
      fileString_tyh = "TyH"+nameFile+"_data_"+time.strftime("%a%d%b%Y-%HH%MM%SS",
        time.localtime())+".data"
75
      ruta1 = path + "/" + str(ahora.year)+str(ahora.month)+str(ahora.day) + "/" +
        folder + "/" +  str(samplesinicio) + "SAMPLESINICIO" + "/" + "dat" + "/" #Ruta
         de salida
77    if not os.path.exists(ruta1): os.makedirs(ruta1)  #Si la ruta no existe, se
        crea
      ruta2 = path + "/" + str(ahora.year)+str(ahora.month)+str(ahora.day) + "/" +
        folder + "/" +  str(samplesinicio) + "SAMPLESINICIO" + "/" + "txt" + "/" #Ruta
         de salida
79    if not os.path.exists(ruta2): os.makedirs(ruta2)  #Si la ruta no existe, se
        crea
      ruta3 = path + "/" + str(ahora.year)+str(ahora.month)+str(ahora.day) + "/"  +
        folder + "/" + str(samplesinicio) + "SAMPLESINICIO" + "/" + "TyH" + "/" #Ruta
        de salida
81    if not os.path.exists(ruta3): os.makedirs(ruta3)  #Si la ruta no existe, se
        crea

83    ruta_fichero = ruta2.strip() + str(fileString)
      ruta_fichero_data = ruta1.strip() + str(fileString_data)
85    ruta_fichero_tyh = ruta3.strip() + str(fileString_tyh)
      return ruta_fichero,ruta_fichero_data,ruta_fichero_tyh
87
  def open_files(ruta_fichero,ruta_fichero_data,ruta_fichero_tyh):
89    f = file(ruta_fichero, "w")
      g = file(ruta_fichero_data, "w")
91    h = file(ruta_fichero_tyh, "w")
      return f,g,h
93
  # Para Martinelli la opcion es 2, para el resto opcion es 1
95 def file_TGS2600(vec_open_valve,succion,heat2600,tendencia,heatPin2600,tiempo,
      switch,algoritmo,ruta,nameFile,nameFile_data1,nameFile_data2,nameFile_tyh,f,
      opcion,tespera,SAMPLESINICIO):

97    f.write ('
        ////////////////////////////////////////////////////////////////\n')
      f.write ('Sensor TGS2600\n')
99    f.write ('Algoritmo: ' + algoritmo + '\n')
      date = time.strftime("%a%d%b%Y-%HH%MM%SS", time.localtime())
101   f.write ('Fecha y hora de inicio: ' + str(date) + '\n')
      f.write ('Ruta del fichero: ' + str(ruta) + '\n')
103   f.write ('Nombre del fichero: ' + str(nameFile) + '\n')
      f.write ('Nombre del fichero de datos 1: ' + str(nameFile_data1) + '\n')
```

```
105   if opcion == 2:
        f.write ('Nombre del fichero de datos 2: ' + str(nameFile_data2) + '\n')
107   f.write ('Nombre del fichero de TyH: ' + str(nameFile_tyh) + '\n')
      f.write ('Electrovalvula (1-METANOL, 2-ETANOL, 3-BUTANOL, 4-AIRE ) \n')
109   f.write ('Conmutacion entre electrovalvulas: ' +str(vec_open_valve)+ '\n')
      f.write ('\n')
111   f.write ('Parametros para la experimetacion, se introducen como argumentos \n')
      f.write ('Succion motor (30-100)  >>> '+ str(succion)+ '%\n')
113
      if opcion == 1:
115     f.write ('Temperatura Promedio TGS2600 (1-100) >>> '+str(heat2600)+'% Pin PWM
        : ' +str(heatPin2600)+ '\n')
        f.write ('La tendencia que va a utilizarse: '+str(tendencia)+'\n')
117
      f.write ('Se ha dejado un tiempo entre captura de gases de: '+str(tespera)+ '
        segundos\n')
119   f.write ('El experimento tiene: ' +str(SAMPLESINICIO)+ ' muestras iniciales\n')
      f.write ('Duracion del experimento: ' +str(tiempo)+ ' minutos o muestras\n')
121   f.write ('El tiempo de switch o conmutacion de las electroválvulas es de: '+str
        (switch)+'\n')
      f.write ('
        ////////////////////////////////////////////////////////////////////\n')
123   f.write ('CAPTURA DE DATOS:\n\n')

125 def cerrar_electrovalvulas():
      GPIO.output(electrovalve1, GPIO.LOW)
127   GPIO.output(electrovalve2, GPIO.LOW)
      GPIO.output(electrovalve3, GPIO.LOW)
129   GPIO.output(electrovalve4, GPIO.LOW)

131 def abrir_electrovalvulas(electrovalvulas):
      print 'Electrovalvulas:' + str(electrovalvulas)
133   for electrovalvula in electrovalvulas:
        GPIO.output(apertura_electrovalvulas[electrovalvula], GPIO.HIGH)
135
    def imprimir_electrovalvulas(elecs):
137
      for elec in elecs:
139     print electrovalvulas[elec]

141 def measure_tyh(h):

143   t = currentThread()
      time.sleep(9)
145
      humidity = 0
147   temperature = 0

149   #Lectura humedad y temperatura
      while getattr(t,"do_run",True):
151     tick_HT = time.time()
        humidity, temperature = DHT.read_retry(sensorTemp22,Temp22,30,1,None)#
        read_retry(sensorTemp22,Temp22)
153     t_HT = time.time()-tick_HT
        instante = datetime.now()
155
        #Cuanto duerme en funcion de lo que tarde en H y T
157     if t_HT > SLEEP_tyh:
          print "Tiempo medicion H y T > SLEEP:", t_HT
159     else:
          print "Tiempo medicion H y T:", t_HT
161       print '\n Sensor DHT22: ' + str(sensorTemp22)
```

```
163         if humidity is not None and temperature is not None:
              print '>>> '+str(instante)+'Temp = ' + str(temperature) + 'Humidity = ' +
        str(humidity) + '\n'
165         else :
              print 'Failed to get reading, Try again!'
167
            wline_h = str(instante) + ' ' + str(temperature) + ' ' + str(humidity) + '\
        n'
169         h.writelines(wline_h)
            h.flush()
171
        tack = time.time()
173     time.sleep(SLEEP_tyh-(tack-tick_HT))

175   return 0

177 def cierre(f,g,h,t):

179   cerrar_electrovalvulas()
      t.do_run = False
181   t.join()
      fecha_fin = datetime.now()
183   print 'Experimento terminado: '+str(fecha_fin)
      f.write('\nEXPERIMENTO FINALIZADO CON EXITO\n')
185   f.write('Fecha y hora de fin de experimentacion:' +str(fecha_fin))
      f.flush()
187   f.close()
      g.close()
189   h.close()
      return
```

codigos/EN_codigo_comun.py

J.3. Código del módulo de representación de datos

```
# coding: utf-8
2 #!/usr/bin/python
import matplotlib.pyplot as plt
4 import re
import os
6
gases = ["Metanol","Etanol","Butanol","Aire"]
8 total_figures = range(1,len(gases)+1)

10 def obtener_ruta(path,ahora,folder,samplesinicio,folder2):
    return path + str(ahora.year)+str(ahora.month)+str(ahora.day) + "/" + folder +
      "/" +  str(samplesinicio) + "SAMPLESINICIO" + "/" + folder2 + "/"
12
def get_data_columns(name,columns):
14
    ret_values = [[] for i in range(len(columns))]
16  with open(name) as f:
      for line in f:
18      line = re.split('[\s\t]',line)
        for i in range(len(columns)):
20        ret_values[i].append(float(line[columns[i]]))

22  return ret_values

24 def plot_vertical_lines(s_ini,switc,n_times,CTE):
```

```
26    cont = s_ini
      plt.axvline(cont,color="black")
28
      for i in range(n_times):
30      if CTE > 0:
          cont+=CTE
32        plt.axvline(cont,color="black")
        cont+=switc
34      plt.axvline(cont,color="black")
      if CTE > 0:
36      cont+=CTE
        plt.axvline(cont,color="black")
38
  def normalizar(x):
40
      miin = min(x)
42    diff = max(x) - miin
      if diff == 0:
44      diff = 1

46    return [(value - miin)*1.0/diff for value in x] #Se multiplica por 1.0 para que
        el valor sea considerado un double.

48  def plot_graf(dictionary,ahora,path):

50    ylabels = ["mV","Resistance","Temperature-grades"]
      plot_forms = ['-r','-b','-g']
52
      for key, value in dictionary.iteritems():
54      switch,s_ini,CTE,ntran,texecution,mexecution,Tmin,Tmax,folder = key
        image_folder = obtener_ruta(path,ahora,folder,s_ini,"images")
56      if not os.path.exists(image_folder): os.makedirs(image_folder)

58      for file in value:
          file_name = re.sub("/","",re.split("[/.]dat",file)[1])
60        xcols = get_data_columns(file,[0,1,2,3])
          x1,x_rest = xcols[0],xcols[1:4]
62
          names_files = ["Plot_sensor_value_"+file_name+".eps","
        Plot_sensor_resistance_"+file_name+".eps","Plot_sensor_temperature_"+file_name
        +".eps"]
64
          #Imprimo los valores del voltaje captados por el sensor, la resistencia
        interna y la temperatura de este.
66        for x_aux,ylabel,nfile,plot_form in zip(x_rest,ylabels,names_files,
        plot_forms):
              plt.plot(x1,x_aux,plot_form)
68            plt.xlabel("Time")
              plt.ylabel(ylabel)
70            plot_vertical_lines(s_ini,switch,ntran,CTE)
              plt.savefig(image_folder+nfile,format="eps")
72            plt.close('all')

74        #Imprimo todo junto
          x2 = normalizar(x_rest[0])
76        x3 = normalizar(x_rest[1])
          x4 = normalizar(x_rest[2])
78        plt.plot(x1,x2,'-r',x1,x3,'-b',x1,x4,'-g')
          plt.xlabel("Time")
80        plt.ylabel("Normalized values")
          plot_vertical_lines(s_ini,switch,ntran,CTE)
82        plt.savefig(image_folder+"Plot_"+file_name+".eps",format="eps")
          plt.close('all')
```

```
84
    return
86
  def plot_elem_grafs(dictionary,ahora,path):
88
    columns = [1,2,3] #Las columnas del voltaje, resistencia y temperatura
90  elems = ['mV','Resistance','Temperature-grades']
    colors = ['azul oscuro','verde','rojo','azul claro','morado','amarillo','negro'
      ]
92
    for key, value in dictionary.iteritems():
94    cont = 0
      switch,s_ini,CTE,ntran,texecution,mexecution,Tmin,Tmax,folder = key
96    image_folder = obtener_ruta(path,ahora,folder,s_ini,"images")
      if not os.path.exists(image_folder): os.makedirs(image_folder)
98
      name_file = []
100   for file in value:
        file_name = re.sub("/","",re.split("[/.]dat",file)[1])
102     xcols = get_data_columns(file,[0]+columns)
        x1,x_rest = xcols[0],xcols[1:4]
104     name_file.append(re.split('[_.]',file_name)[2])

106     for column in columns:
          plt.figure(column)
108       plt.plot(x1,x_rest[column-1],'-')

110   for column,elem in zip(columns,elems):
        v = plt.figure(column)
112     plot_vertical_lines(s_ini,switch,ntran,CTE)
        plt.xlabel("Time")
114     plt.ylabel(elem)
        plt.savefig(image_folder+"Plot_sensor_"+elem+"_"+str(cont)+".eps",format="
      eps")
116     plt.close(v)

118   f = open(image_folder+"Plot_sensor_"+str(cont)+"_legend.txt","w")
      for i in range(len(name_file)):
120     f.write(name_file[i]+" "+colors[i%len(colors)]+"\n")
      f.close()
122

124   cont+=1

126 return

128 def plot_elems_div_graf(dictionary,ahora,path):

130 plt.figure(1)
    figures = [1,1,2,2,3,3,4,4,2,1,3,1,4,3,2,4]
132 subplotordes =
      [221,222,222,223,223,224,224,222,221,223,221,224,223,222,224,221]
    columns = [1,2,3] #Las columnas del voltaje, resistencia y temperatura
134 elems = ['mV','Resistance','Temperature-grades']
    colors = ['azul oscuro','verde','rojo','azul claro','morado','amarillo','negro'
      ]
136
    for key, value in dictionary.iteritems():
138   cont = 0
      switch,s_ini,CTE,ntran,texecution,mexecution,Tmin,Tmax,folder = key
140   image_folder = obtener_ruta(path,ahora,folder,s_ini,"images")
      if not os.path.exists(image_folder): os.makedirs(image_folder)
142
```

```
    if CTE > 0:
144   ini = s_ini+CTE+switch+CTE
    else:
146   ini = s_ini + switch
    fin = ini + switch
148
    for column,elem in zip(columns,elems):
150   for file in value:
        xcol = get_data_columns(file ,[column])[0]
152     for fig,sub in zip(figures,subplotordes):
          plt.figure(fig)
154       plt.subplot(sub)
          plt.plot(range(1,(switch+1)),xcol[ini:fin],'-')
156
          if CTE > 0:
158         ini = fin + CTE
          else:
160         ini = fin
          fin+=switch
162
        if CTE > 0:
164       ini = s_ini+CTE+switch+CTE
        else:
166       ini = s_ini + switch
        fin = ini + switch
168
      for g,f in zip(gases,total_figures):
170     v = plt.figure(f)
        plt.savefig(image_folder+"Plot_sensor_"+g+"_"+elem+"_transitions_"+str(
    cont)+".eps",format="eps")
172     plt.close(v)

174   f = open(image_folder+"Plot_sensor_div_graf_"+str(cont)+"_legend.txt","w")
    for i in range(len(value)):
176   f.write(re.split('[_.]',re.sub("/","",re.split("[/.]dat",value[i])[1]))[2]+
    " "+colors[i%len(colors)]+"\n")
    f.close()
178
    cont+=1
180
  return
182
def martinelli_vertical_separation(time_total,mexecution,switch,CTE,sini,n_times)
    :
184
  ini = 0
186 fin = sini*16
  cont = sum(time_total[ini:fin])
188
  ini = fin
190 fin+=16

192 print mexecution,switch,CTE,sini,n_times
  if mexecution == 1:
194   for i in range(n_times):
      if CTE > 0:
196     while sum(time_total[ini:fin]) < CTE:
          fin+=16
198     cont+=sum(time_total[ini:fin])
        plt.axvline(cont,color="black")
200     ini = fin
        fin+=16
202
```

```
      while sum(time_total[ini:fin]) < switch:
        fin+=16

      cont+=sum(time_total[ini:fin])
      plt.axvline(cont,color="black")
      ini = fin
      fin+=16
  else:
    pos = sini-1
    plt.axvline(time_total[pos],color="black")

    for i in range(n_times):
      if CTE > 0:
        pos+=CTE
        plt.axvline(time_total[pos],color="black")

      pos+=switch
      plt.axvline(time_total[pos],color="black")

    if CTE > 0:
      pos+=CTE
      plt.axvline(time_total[pos],color="black")

  return

def obtener_tiempo_pulsos_martinelli(time_total,switch,CTE,s_ini,ntran):

  ini = 0
  fin = 16
  result = []

  for i in range(s_ini+(ntran+CTE)*switch+CTE):
    result.append(sum(time_total[ini:fin]))
    fin+=16

  return result

def martinelli_pulses(time_up,time_down,ini,fin):

  ret_values = [0]

  for i in range(ini,fin):
    ret_values.append(sum([time_up[i],ret_values[len(ret_values)-1]]))
    ret_values.append(sum([time_down[i],ret_values[len(ret_values)-1]]))

  ejey = [3.3,0]*(len(ret_values)/2)
  if len(ret_values)%2 == 1:
    ejey.append(3.3)

  return ret_values,ejey

def martinelli_temp_pulses(time_total,temp,texecution,ini,Tmin,Tmax):

  x_time = []
  y_temp = []
  if texecution == 1:
    bucle = len(time_total[ini:])/8
    x_time.append(0)
    fin = 8

    for i in range(bucle):
      x_time.append(sum(time_total[ini:fin]))
      fin+=8
```

```
266
      y_temp=[Tmax,Tmin]*(len(x_time)/2)
268     if len(x_time)%2 == 1:
          y_temp.append(Tmax)
270   else:
      bucle = len(time_total[ini:])
272     x_time.append(time_total[ini])
      y_temp.append(temp[ini])
274
      for i in range((ini+1),bucle):
276       x_time.append(sum([x_time[len(x_time)-1],time_total[i]]))
        y_temp.append(temp[i])
278
    return x_time,y_temp
280
  def plot_martinelli(dictionary,ahora,path):
282
    for key, value in dictionary.iteritems():
284     switch,s_ini,CTE,ntran,texecution,mexecution,Tmin,Tmax,folder = key
      image_folder = obtener_ruta(path,ahora,folder,s_ini,"images")
286     if not os.path.exists(image_folder): os.makedirs(image_folder)

288     for file in value:

290       fig,ax1 = plt.subplots()
        ax2 = ax1.twinx()
292       x = get_data_columns(file,[1,2,3,4])
        time_up = x[0]
294       time_down = x[1]
        time_total = x[2]
296       temp = x[3]

298       pulses,ejey = martinelli_pulses(time_up,time_down,0,len(time_up))
        time,temp = martinelli_temp_pulses(time_total,temp,texecution,0,Tmin,Tmax)
300
        ax1.plot(pulses,ejey,drawstyle='steps-pre',color="b",fillstyle='full',
      markeredgewidth=0.0)
302       if texecution == 1:
          ax2.plot(time,temp,drawstyle='steps-pre',color="r",fillstyle='full',
      markeredgewidth=0.0)
304       else:
          ax2.plot(time,temp,'r.-')
306       martinelli_vertical_separation(time_total,mexecution,switch,CTE,s_ini,ntran
      )
        plt.savefig(image_folder+"Plot_"+re.sub("/","",re.split("[/.]dat",file)[1])
     +"N.eps",format="eps")
308       plt.close('all')

310   return

312 def martinelli_ventana_tiempo(time_total,switch,CTE,sini,Tmin,Tmax):

314   ini = sini*16
    fin = ini+16
316
    temps = []
318   times = []
    time = [0]
320
    #Leo el aire innecesario y la primera muestra, no la necesito
322   if CTE > 0:
      while sum(time_total[ini:fin]) < CTE:
324       fin+=16
```

```
      ini=fin
326   fin+=16

328   while sum(time_total[ini:fin]) < switch:
        fin+=16
330   ini=fin
      fin+=16
332
      for i in range(16):
334     #Leo, si hay aire, las muestras del aire
        if CTE > 0:
336       while sum(time_total[ini:fin]) < CTE:
            fin+=16
338       ini=fin
          fin+=16
340
        #Voy leyendo los gases
342     while sum(time_total[ini:fin]) < switch:
          fin+=16
344
        if sum(time_total[ini:fin]) == switch:
346       fin_aux = fin
        else:
348       fin_aux = fin - 16

350     if ini == fin_aux:
          if sum(time_total[ini:(fin_aux+8)]) < switch:
352         time.append(sum(time_total[ini:(fin_aux+8)]))
          time.append(switch)
354     else:
          for j in range(1,((fin_aux-ini)/8)+1):
356         time.append(sum(time_total[ini:(ini+8*j)]))

358       if sum(time_total[ini:(fin_aux+8)]) < switch:
            time.append(sum(time_total[ini:(fin_aux+8)]))
360
          if sum(time_total[ini:fin_aux]) < switch:
362         time.append(switch)

364     ini=fin
        fin+=16
366     times.append(time)
        temp=[Tmax,Tmin]*(len(time)/2)
368     if len(time)%2 == 1:
          temp.append(Tmax)
370     temps.append(temp)
        temp = []
372     time = [0]

374   return times,temps

376 #Imprime en subplots, los valores que se han capturado con un numero de pulsos
    def martinelli_numero_pulsos(time_total,argtemp,switch,CTE,sini):
378
      ini = sini*16
380   fin = ini+16

382   temps = []
      times = []
384
      if CTE > 0:
386     for i in range(1,CTE):
          fin+=16
```

```
388       ini=fin
          fin+=16
390
      for i in range(1,switch):
392       fin+=16
      ini=fin
394   fin+=16

396   for i in range(16):
        if CTE > 0:
398       for j in range(1,CTE):
            fin+=16
400       ini=fin
          fin+=16
402
        for j in range(1,switch):
404       fin+=16

406     time=[0]
        temp=[argtemp[ini]]
408
        for j in range((ini+1),fin):
410       time.append(sum([time[len(time)-1],time_total[(j-1)]]))
          temp.append(argtemp[j])
412     ini=fin
        fin+=16
414     times.append(time)
        temps.append(temp)
416
      return times,temps
418
#Funcion que imprime las graficas segun el modelo elegido
420 def martinelli_temp_pulses_div_graf(time_total,argtemp,switch,mexecution,CTE,sini
      ,Tmin,Tmax):

422   if mexecution == 1:
        return martinelli_ventana_tiempo(time_total,switch,CTE,sini,Tmin,Tmax)
424   else:
        return martinelli_numero_pulsos(time_total,argtemp,switch,CTE,sini)
426
#Imprime los valores de las graficas en distintos subplots, segun los gases que
      se analizan
428 def plot_martinelli_div_graf(dictionary,ahora,path):

430   figures = [1,1,2,2,3,3,4,4,2,1,3,1,4,3,2,4]
      subplotordes =
        [221,222,222,223,223,224,224,222,221,223,221,224,223,222,224,221]
432
      for key, value in dictionary.iteritems():
434     switch,sini,CTE,ntran,texecution,mexecution,Tmin,Tmax,folder = key
        image_folder = obtener_ruta(path,ahora,folder,s_ini,"images")
436     if not os.path.exists(image_folder): os.makedirs(image_folder)

438     for name in value:
          ret = get_data_columns(name,[3,4])
440       times,temps = martinelli_temp_pulses_div_graf(ret[0],ret[1],switch,
        mexecution,CTE,sini,Tmin,Tmax)
          for fig,sub,time,temp in zip(figures,subplotordes,times,temps):
442         plt.figure(fig)
            plt.subplot(sub)
444         plt.xlabel("Time (seconds)")
            plt.ylabel("Temperature %5V")
446         if texecution == 1:
```

```
            plt.plot(time,temp,drawstyle='steps-pre',fillstyle='full',
    markeredgewidth=0.0)
448         else:
              plt.plot(time,temp,'.-')
450
    for g,f in zip(gases,total_figures):
452   v = plt.figure(f)
      plt.savefig("Plot_martinelli_sensor_"+g+"_transitions.eps",format="eps")
454   plt.close(v)
```

codigos/plot.py